**分类号**  **学号**

**学校代码 密级**

****

硕士学位论文

**题目：基于Web的增材制造预处理平台的设计与实现**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **学位申请人** | **：** |  | **张鹏** |  |  |
| **学科专业** | **：** |  | **计算机技术** |  |  |
| **指导教师** | **：** |  | **李国宽副教授** |  |  |
| **答辩日期** | **：** |  | **2021年6月XX日** |  |  |

**A Dissertation Submitted in Partial Fulfillment of the Requirements**

**for the Professional Master Degree**

**Design and implementation of additive manufacturing pretreatment platform based on Web**

**Candidate : Peng Zhang**

**Major : Computer technology**

**Supervisor : Prof. Guokuan Li**

**Huazhong University of Science and Technology**

**Wuhan 430074, P. R. China**

**June, 2021**

**独创性声明**

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期： 年 月 日

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□， 在 年解密后适用本授权书。

本论文属于

不保密□。

（请在以上方框内打“√”）

学位论文作者签名： 指导教师签名：

日期： 年 月 日 日期： 年 月 日

# 摘 要

增材制造（Additive Manufacturing, AM）俗称3D打印，是一种以数字模型文件为基础，通过软件与数控系统将材料进行累加制造的方法。随着“中国制造2025”战略的提出，新的制造业将是汇集物联网、云制造、人工智能等前沿技术的大体系。针对3D打印最关键的数据处理部分，目前主要可分为开源软件和商业专用软件两类，开源软件大多只是对国外开源软件进行汉化处理，无法根据自己需求进行优化；商业专用软件则价格昂贵且操作复杂，不利于3D打印技术的普及和大众化发展。并且许多数据预处理软件载体都是本地客户端，操作完全离线，无法对数据文件进行统一管理，类似的模型处理过程没有整合封装，不利于3D打印产品的个性化定制与平台化应用发展。而随着Web3.0的时代到来，网速的快速增长，浏览器引擎的处理速度飞速提升，实时三维数据渲染的方案越来越完善，为3D打印预处理平台的构建打下了良好的基础。因此，本文通过对3D打印数据预处理过程中涉及到的流程与数据存储形式进行研究和分析，结合Web应用的优势，将打印过程分为几个核心模块，并自主开发出一个基于B(Browser)/S(Serve)模式的在线模型数据预处理应用。

首先，介绍了目前3D打印的软件需求以及行业发展现状，并根据打印流程分析，将数据处理分为几个关键模块，并简要介绍了Web应用的整个架构和全栈开发知识体系。其次，详细地阐述了系统中的数据结构，以及处理过程中涉及到的几个核心算法，采用前后端分离的开发模式实现了对系统用户和模型数据的管理，还有导入导出三维模型文件、分层切片、轨迹规划和GCode生成的功能。最后，部署在云服务器上，且在不同配置的电脑上对该网页应用进行试验验证。试验结果表明，增材制造数据预处理流程可以与Web应用良好地结合，用户交互界面合理，操作可行性强，算法执行效率高，基本上可以满足3D打印的数据处理需求，并且具备较高的扩展性和可移植性。

**关键词：**3D打印；Web开发；STL模型；平台应用；前后端分离

# Abstract

英文摘要字体为Times New Roman，小四，1.5倍行距。

英文摘要和关键词应与中文相对应。英语摘要用词应准确，使用本学科通用的词汇；摘要中主语（作用）常常省略，因而一般使用被动语态；应使用正确的时态，并要注意主、谓语的一致，必要的冠词不能省略。

**Key words:** Optical microscopy; Brain; Neuron

**目** **录**

[题目：基于Web的增材制造预处理平台的设计与实现 I](#_Toc67587255)

[A Dissertation Submitted in Partial Fulfillment of the Requirements II](#_Toc67587256)

[for the Professional Master Degree II](#_Toc67587257)

[摘 要 I](#_Toc67587258)

[Abstract II](#_Toc67587259)

[1. 绪论 1](#_Toc67587260)

[1.1 课题背景及研究意义 1](#_Toc67587261)

[1.2 国内外研究现状 2](#_Toc67587262)

[1.3 本文研究内容及安排 9](#_Toc67587263)

[2. 增材制造预处理Web平台的关键技术及原理 12](#_Toc67587264)

[2.1 系统整体概述 12](#_Toc67587265)

[2.2 涉及关键技术 13](#_Toc67587266)

[2.3 本章小结 30](#_Toc67587267)

[3. 系统需求分析及概要设计 32](#_Toc67587268)

[3.1 需求分析 32](#_Toc67587269)

[3.2 系统用户及用例 32](#_Toc67587270)

[3.3 系统业务流程分析 34](#_Toc67587271)

[3.4 系统详细功能模块设计 35](#_Toc67587272)

[3.5 本章小结 39](#_Toc67587273)

[4. 增材制造预处理Web平台实现与测试 40](#_Toc67587274)

[4.1 开发环境搭建 40](#_Toc67587275)

[4.2 前端实现 42](#_Toc67587276)

[4.3 模型交互实现 53](#_Toc67587277)

[4.4 增材制造预处理实现 58](#_Toc67587278)

[4.5 数据库设计与实现 67](#_Toc67587279)

[4.6 服务端设计与实现 68](#_Toc67587280)

[4.7 系统测试 71](#_Toc67587281)

[4.8 本章小结 82](#_Toc67587282)

[5. 总结与展望 83](#_Toc67587283)

[5.1 本文主要内容及结论 83](#_Toc67587284)

[5.2 本文的主要创新点 83](#_Toc67587285)

[5.3 展望 84](#_Toc67587286)

[致 谢 85](#_Toc67587287)

[参考文献 86](#_Toc67587288)

目录编写要求：

1. 目录自动生成时，只需列出二级标题；
2. 中文宋体，小四，行距：1.5倍；
3. 注意避免出现不具辨识度的标题：如材料与方法、实验结果、分析与讨论、结论，若属于这种类型的论文时，可以把结果部分拆开，根据结果拟定题目

# 绪论

## 课题背景及研究意义

中国近十一年来一直位列全球制造大国之首，因此制造业是事关国民经济命脉的主要支柱产业，代表了国家的科技发展与综合实力水平。2015年两会上初次提出“中国制造2025”的重要战略规划与目标，未来的制造业无疑是要结合物联网、云制造、人工智能等前沿信息技术，朝着智能制造的方向发展。

3D打印又被称为增材制造、积层制造，是一种以数字模型文件为基础，运用粉末状金属或塑料等可粘合材料，通过逐层打印的方式来构造物体的技术，该技术最早提出于20世纪80年代。常用于模具制造、工业设计等领域，后逐渐用于一些产品的直接制造，与传统加工工艺对比起来，增材制造加工效率高，耗材低，易于成型，且可以满足定制化需求，是制造业中比较颠覆性的技术。目前，根据打印的工艺和成型材料，可以将3D打印分为立体光刻、熔融沉积成型、立体平板印刷、选择性激光烧结、电子束自由成形制造和分层实体制造等方法[1]。3D打印的过程可以由离散和堆积两大部分组成：离散是使用一系列的平面对数据模型进行切片处理，获得每一层平面的轮廓；分层切片后根据不同的工艺要求，对每个轮廓进行计算处理，得到模型每一层的加工路径，再将加工路径翻译成打印机器可执行的加工代码，使得机器可以根据生成的代码进行加工，这就是堆积的过程。

在3D打印整个流程中最核心的部分就是对模型数据的处理，不同的参数以及计算算法都将直接影响产品的质量和精度。针对模型的数据处理就要涉及到模型导入、模型拓扑重建、模型切片、扫描路径生成等步骤，在实际生产过程中可能要使用到多个软件，并且大多数开源软件都是一种黑盒模式，无法进行自适应优化处理，同一个模型使用不同软件得到的打印代码都是有偏差的，因此无法满足专业人士的需求，而且处理过程的繁琐也不利于3D打印技术面向大众进行推广。

众所周知，自二十一世纪以来发展最快的就是互联网行业，以互联网为载体的信息技术已经逐渐改变传统生产及商业模式，引发全球工业产业的技术升级。尤其在近几年基于浏览器的应用蓬勃发展，每年的在线应用都是成倍增长，甚至客户端软件都开始逐步开发Web端应用，比如CAD绘图，Web版Photoshop以及在线视频剪辑等。所以研究如何将3D打印流程结合Web应用，使得操作精简化、计算统一化、数据共享化、功能平台化，这将具有十分重要的现实意义。

## 国内外研究现状

### 3D打印技术研究现状

3D打印与传统制造工业相比，最直观的感受就是它不需要模具，而是利用计算机建模数据，就可以通过逐层累加的方式进行填充打印。相较于传统的材料去除（切削加工）技术，是一种“自上而下”材料累加的制造方法。自20世纪80年代末增材制造技术逐步发展，期间也被称为“材料累加制造”(Material increase manufacturing)、“快速原型”(Rapid proto-typing)、“分层制造”(Layered manufacturing)、“实体自由制造”(Solid free－Form fabrication)等[4]。美国材料与试验协会国际委员会对增材制造和 3D 打印有明确的概念定义。3D 打印是指采用打印喷头或其他打印技术沉积材料来制造物体的技术，3D 打印也常用来表示“增材制造”技术[5]。

根据打印材料、形成模型方式不同，3D打印方法主要分为六类，分别是：熔融沉积成形（Fused Deposition Modeling，简称为FDM）、金属激光熔融沉淀（Laser Metal Deposition，简称为LMD）、光固化几何体成形（Stereo lithography Apparatus，简称为SLA）、选择性激光烧结（Selective Laser Sintering，简称为SLS）、分层实体制造（Laminated Object Manufacturing，简称为LOM）、电子束熔融（Electron Beam Melting，简称为EBM）[6]。其中最常见的3D打印技术就是FDM技术，FDM技术最早是由美国的Scott Crump博士于1988年研制成功，它具有材料成本低、利用率高、成形速度快、设备小巧、操作简单等特点[3]。该技术适用于大多数复杂模型的制造，被广泛应用在建筑模型设计、医疗器械制备、航空机械零件生产等领域。

FDM技术的主要流程是，先通过建模软件得到满足需求的模型文件，再将得到的模型文件通过专业软件进行分层切片处理，根据切片得到的轮廓判定轮廓之间的内外关系，再根据轮廓形成的不同包络区域进行路径规划填充，最后将规划路径转为打印机可执行的G代码。打印机读取GCode之后，加热打印喷头，将融化的材料匀速挤出，沿着每个切片平面方向进行均匀涂抹，直至不断调整高度填充整个模型，就生成了最终的打印形件。FDM打印机结构如图 1‑1所示：



图 1‑1 FDM打印机结构示意图

3D打印的基础流程包括以下五个方面：

1. 三维模型建模：目前市场上三维模型制作方式有很多种，比如人工建模，扫描建模，照片建模等等，根据不同的建模方式选择的软件也有所不同，如：CAD、3DMax、Zbrush、SketchUp等。三维模型生成方式十分灵活，还可以添加渲染、特效、特殊材料等，满足用户的个性化需求。
2. STL文件格式转换：常见的三维文件格式有：ABC（动画模型文件格式）、glTF（动画模型文件格式）、FBX（3D文件格式）、BVH（人体特征动画文件格式）、OBJ（标准3D模型文件格式）、DAE（纯文本模型格式）、STL（三角网格文件格式）、3DS（早期3D文件格式）等，不同的文件格式适用的场景不一样，并且现在市面上有许多软件都可以将不同格式的文件进行互相转换，由于本文研究基于三角面片网格模型，所以我们选择的是STL格式的三维模型。
3. 分层处理：模型在结构上是由一个个三角面片组成，所以需要对STL文件进行分层切片，将同一层相邻的面片顺序存储在每个轮廓的数据结构中，分的层数越多，模型精度就越高，但是耗费的计算时间以及计算空间就越大。
4. 路径规划：分层处理后，将得到各个轮廓的数据集，需要遍历轮廓找出内外轮廓关系，划分不同的填充区域，还要根据打印机特性设置不同的填充方式，保证有效合理地去生成打印轨迹。
5. 打印成品：打印机根据生成的GCode文件，控制打印针头沿着预设的轨迹、速度、喷丝方向进行堆积喷涂，直至最终形成三维实体器件。

在打印的过程中，数据信息流的传递方式如图 1‑2所示：



图 1‑2 3D打印过程数据信息流示意图

预处理的过程通常就需要用到专业的三维数据处理软件，传统的商业计算机三维设计软件大致可以分为三类。一类是面向工业产品设计制造的计算机辅助设计系统（Computer-Aided-Design）系统，另一类是主要用于电影、动画、游戏等行业的计算机图形系统（Graphic），最后一类就是面向3D打印的制造应用三维模型设计系统[7]。

从1960年代Ivan Sutherland博士提出的SketchPad[8]系统开始，计算机辅助设计系统已经有50多年历史了。在1970和1980年代，Rochester大学的Herb Voelcker等人开展了大量对CSG（Constructive Solid Geometry）建模技术的研究[9-11]，并研发出了PADL建模器；剑桥大学的Ian Braid和Charles Lang等人大力发展了BRep（Boundary Representation）建模技术[12-14]，导致了ACIS等建模器的诞生。此后SolidWorks公司和AutoDesk公司分别于1995年和1999年发布了SolidWorks设计系统和Inventor设计系统，这两者就成为计算机三维辅助设计软件的首要选择。而随着3D打印技术的逐渐热门，市场上开始推出一些专门为3D打印而开发的三维建模软件，比如：Cura，CraftWare，Simplify3D等[15]。这些软件都有着直观、快速、易用等特点，适合初学者上手，并且有着模型支撑设置以及丰富的预览功能，使得专业用户也常常会选择这些软件。

而国内3D打印软件的研究始于90年代,不同机构根据成形原理、打印材料和加工工艺等方面做出了一些比较成形的软件[16]。尽管国内研究起步晚，但是也有一些科研团队在3D打印领域处于领先地位，比如华中科技大学的史玉升教授，已开发出多款能打印复杂模型的3D打印机，面向的是高端制造业，产品价格极高。而对于大众化的3D打印市场，国内暂时只有数十家企业在从事经济型3D打印机的制造，而这些3D打印机软件都是利用国外开源软件进行汉化处理，对于核心算法无法进行改进，利用开源软件去进行商业化使用也容易涉及到知识产权的问题，所以如何将核心技术掌握在自己手里，并且将其作为一个产业化发展去考虑与实践，这是将来国内3D打印行业需要突破的地方。

### Web技术研究现状

1989年欧洲核子研究中心实验室研究员蒂姆·伯纳斯·李（Tim Berners-Lee）发明了万维网[17]，是互联网史上具有划时代意义的事件，自此之后Web应用开始爆炸式的繁荣起来。下面是Web发展的几个里程碑事件：

1. HTML（Hyper Text Markup Language）与CSS（Cascading Style Sheets）：HTML是由万维网发明者Tim Berners-Lee和同事Daniel W. Connolly于1990年创立的一种标记语言[18]。它通过标记式的指令，将影像、声音、图片、文字动画、影视等内容显示出来；CSS在1994 年由Håkon Wium Lie 提出，主要给网页提供页面装饰。最初版本是Håkon Wium Lie与设计 Argo 浏览器的Bert Bos一起创造的，并在1996 年 12 月，由W3C 推出了 CSS 规范的第一版本[19]。

2. Mosaic浏览器：Mosaic是互联网历史上第一个普遍使用和能够显示图片的网页浏览器，在1993年发表，并于1997年1月7日正式终止开发和支持[20]。由于是第一款具有方便易用的图形界面的浏览器，让当时许多人了解到了Internet，尽管后来停止了前进的脚步，但是它对后来出现的浏览器影响深远。

4. JavaScript：JavaScript是一种基于客户端浏览器的网页脚本语言，它可以直接和HTML文档整合，其编译过程不需要专门的编译器，通过浏览器进行逐行解释和执行[23]。最早是由网景公司（Netscape）发明，主要目的是想用一个脚本语言来解决一些表单的验证问题，而真正开始影响到Web开发进程的事件是1999年IE4.0发布，内置异步通讯的组件，允许脚本语言发送HTTP请求到服务器，这就是Ajax的原型。

5. Ajax（Asynchronous Javascript And XML）：Ajax是指一种创建交互式、快速动态网页应用的网页开发技术，无需重新加载整个页面，而能够更新部分网页的技术[21]。Ajax于1998年前后开始应用，在这之前，网页交互方式是用户点击一个按钮后，需等待漫长的数据与服务器交互，期间用户无法进行任何操作，而Ajax可以让应用在进行请求交互时，用户不必等待结果，而是能对页面做其它操作，极大提高了用户体验，也为前后端分离的开发模式奠定了基础。

6. Flash：Flash 最早期的版本称为Future Splash Animator，是1996年11月，乔纳森·盖伊(Jonathan Gay)和他的六人小组编写而成[22]。经过多年的开发迭代，于2005年9月，Flash 8.0发布，增强了对视频的支持，可将其打包成flv文件，同年，Adobe耗资34亿美元并购Macromedia，Flash正式成为Adboe公司旗下的产品。由于可以在浏览器中播放动画，Flash开始异军突起引爆了整个流媒体行业，并在2007年左右，随着Flash 的交互技术进步，网页游戏的浪潮开始席卷全球，在2010年Flash达到巅峰，直至2015年HTML5的兴起和移动互联网的发展，再加上Flash本身的安全性问题，Flash才渐渐地退出历史舞台。

7. jQuery：jQuery是2006年发布的一个免费开源并且跨浏览器的JavaScript框架,其核心设计理念是写更少的代码，做更多的事情（Write Less Do More）[24]。jQuery最大的优点就是减少了开发者对页面DOM树的维护，封装了许多JavaScript处理函数以及Ajax异步调用实现的封装，并且都做了兼容性处理，大大减小了JavaScript操作页面的复杂度，使得JavaScript 代码和 HTML 代码完全分离，便于代码的维护和修改。

8. Google Chrome：Google浏览器，是一款由Google公司在2008年开发的网页浏览器，Chrome基于webkit内核。由于良好的设计标准和市场反应，促进浏览器快速迭代，直接撼动了IE在浏览器界的地位。由于它便捷的调试工具以及丰富的插件市场，受到了广大Web程序开发者的喜爱。

10. Bootstrap：2011年Twitter开源的网页GUI框架，作用就好比jQuery对JavaScript进行了浏览器兼容，那么Bootstrap则是兼容了不同浏览器的CSS部分[25]。它带来的好处就是你无需知道CSS，无需前端，只要根据文档说明，就可以快速搭建起一个美观的网页，让开发者重心放在页面的逻辑交互，而不是处理各个浏览器的样式兼容，虽然它会导致页面的千篇一律，但是它为了解决响应式问题所提出的栅格化系统，是今后各式各样的UI框架借鉴的鼻祖。

12. Node.js：由于V8引擎的出现，满足了对高性能Web服务器的想象，于是在2009年5月由Ryan Dahl开发，基于Chrome V8引擎的JavaScript运行环境就此诞生[26]。设计模式为事件驱动、非阻塞式I/O，可以让JavaScript 运行在服务端的开发平台，后面在Node.js的基础上又推出了NPM包管理器，让大量工具包得以共享使用，推动着前端行业的增量式发展。

13. HTML5+CSS3：2014年10月HTML5规范制定完成，HTML5相比于HTML而言，增添了许多标签，可以更多维度的构建出对用户有价值的数据。HTML5的APP Cache以及本地存储功能大大缩短了一些APP的启动时间，同时，HTML5通过将内部和外部的数据直接连接，有效的解决了设备之间的兼容性问题，此外，HTML5具有动画、多媒体模块、三维特性等等，可以替代部分Flash和Silverlight能够实现的功能，并且具有更好的处理效率[27]。CSS3则是在CSS的基础上，采用模块化开发结构包括框模型、动画模型、背景与边框模型、文字效果、列表展示、多列布局等等，还补充了HTML5中的文本样式[28]。对于开发者来说，可以用更简洁的代码，得到更丰富的页面效果，大大地提高了工作效率。

14.前后端分离模式：早期网站的服务模式是发送一个http请求，然后服务器对请求做完处理，最后将静态的资源返回到页面中，因此所有静态资源和业务代码统一部署在同一台服务器上，整个流程如图 1‑3所示：



图 1‑3 早期网站服务流程

在这样的模式下，前端工作就只有将UI设计师提供的原型图还原实现成静态页面交付于后端工程师，然后后端人员再加入逻辑代码，使得前后端工作极度耦合，相互依赖严重，一旦设计改变，就需要重新走一遍流程，导致开发效率低，代码难以维护。直到2013年左右，前后端分离思想的出现，基于Node.js服务器，让前端去负责View层和Controller层，后端只负责Model层和Service层，由于Node.js中间层的加入，浏览器不再直接请求服务端的API，而是先去请求Node.js，由Node.js对服务端的API发送HTTP请求，Node.js收到服务端返回的JSON格式数据再去渲染HTML页面[29]。前后端分离不仅将开发人员的职责明确区分开来，还大大提升了适配性，如果Controller层掌握在后端手里，那么多端适配的时候，后端就要维护不同环境下的页面，导致模板无法重用，而且大多数时候后端的业务逻辑是一样的，如果前后端无法分离，就多出了很多无意义的工作量。

15.WebGL（Web Graphics Library）：WebGL是一种用于Web的标准3D绘图协议，基于OpenGL ES2.0的JavaScript API进行开发。OpenGL（Open Graphics Library）是一种成熟的跨平台高性能3D渲染API，它由Silicon Graphics Inc.开发，然后在 1992年发布，目前由The Khronos Group管理。它是一种基于渲染管线、独立于硬件，且具有客户端-服务器结构化的API[30]。WebGL初始版本是于2011年3月被提出，WebGL2.0的规范发展于2013年并于2017年1月完成。自WebGL问世后，网页应用又可以开拓出更多可能性，例如3D页游，虚拟现实等。

Web技术经过这二三十年来的发展，已经有了天翻地覆的改变，从最初只能制作一个单纯的静态文本页面，到如今可以承载Web OS、AR、VR和人工智能等前沿科技领域的强大技术。截止到2020年底，全球网站数量已经突破十亿，尤其是近十年来，网页数量呈几何级数增长，各式各样的应用都在朝着Web化的方向发展，因为这不仅仅是将应用迁移到Web端，而是思维开始向云计算、云服务、平台化等潮流转变。当底层硬件性能的壁垒不断被打通，Web几乎是未来应用载体的首要选择。

## 本文研究内容及安排

### 本论文的研究内容

随着Web进入3.0时代以后，Web端实时通信、大数据流传输以及WebGL技术的日趋成熟，我们可以看到Web App正以势不可挡的趋势发展，快速占领各个行业，很多基于客户端的程序都在逐渐向Web端扩展。比如：Web Photoshop，AutoCAD Web App、Tim在线文档等产品。而增材制造预处理过程中所用到的软件都是离线的客户端程序，每个流程的数据信息都是独立使用，没有统一的数据存储，对于每一次的处理记录都无法做到数据追踪，并且增材制造预处理过程中的算法也会不断优化更新，而客户端程序的更新要面临多端适配的问题，并且具有版权的算法在客户端程序里也会面临着被破解的风险，而Web化的好处不仅是无感知更新，即开即用等优点，并且还能将计算云服务化，数据云存储共享化，这是未来互联网应用发展的必然趋势。在五六年前Web应用程序的执行效率确实无法超过客户端，但是随着技术的不断发展，曾经的技术瓶颈被一个个打破，浏览器厂商与行业标准在不断进步趋同，因此基于Web的增材制造预处理平台的实现将具有十分重要的意义。

基于以上原因，本课题对Web技术如何应用到增材制造预处理过程进行了研究，主要研究内容有以下几点：

1. 本文设计并实现基于Web的增材制造预处理系统，是一个旨在服务3D打印学习者并对用户产生的数据进行合理管理的平台，是作为一个3D打印软件服务行业的扩展形态而存在。
2. 对WebGL技术和Three.js框架的概念、优缺点及发展现状进行了调研与系统性总结，基于开放的底层Api对系统中模型的展示与交互操作进行二次封装，使得模型交互更加合理，提高用户体验。
3. 针对当前应用给出了合理的技术选型，并完成Web前端、NodeJS后端以及数据库的搭建，并对整个应用架构进行了组合设计，对于系统开发所涉及到的功能模块进行了二次封装，提升整个应用代码的可阅读性。
4. 将增材制造预处理过程中涉及到的几大算法进行研究分析，详细阐述了每个算法的流程，并且根据实际Web应用开发的数据结构，对算法进行了优化调整，并用实际模型对算法进行了测试。
5. 完成平台实现，确定整个系统的界面、功能以及交互方式，实现了用户登录注册、模型上传修改、模型导入导出、模型放缩还原、模型切片、轨迹填充、GCode生成和动画演示等功能，并截取主要代码进行辅助说明。
6. 将应用程序部署在服务器上，并在不同的设备上对系统进行测试分析。

### 本论文的结构安排

第一章，绪论。主要介绍本文课题的研究背景与研究意义，3D打印技术的研究现状，Web技术的研究现状。

第二章，增材制造预处理Web平台的关键技术及原理，首先对系统进行整体概述，然后介绍实现系统涉及到的关键技术，主要由增材制造预处理相关算法和Web应用全栈化开发相关技术两部分组成。

第三章，系统需求分析及概要设计，本章对系统的应用功能进行了需求分析，根据系统需求来对系统进行用例分析，然后对系统的业务流程做阐述，并对每个功能模块进行详细设计。

第四章，增材制造预处理Web平台实现与测试，本章从开发环境搭建、前端实现、模型交互实现、增材制造预处理实现、数据库设计和服务端设计这几个部分出发对每个模块的具体实现细节进行了阐述，然后展示了相应实现的效果图，最后对系统进行测试与结果分析。

第五章，总结与展望，概述本文研究的主要内容，叙述本文的主要创新点，并提出系统研究实现中遇到的问题和不足，并对基于Web的增材制造预处理平台进行了展望。

# 增材制造预处理Web平台的关键技术及原理

## 系统整体概述

目前市面上的预处理软件都是客户端软件，并且其中核心算法的版权是属于国外开发人员，因此无论是根据实际情况去修改代码，还是出现Bug以后需要更新软件，都需要获取相应的开源资源，这就导致单一的软件无法满足用户的需求，需要将软件组合起来使用。还有不同的用户都拥有不同的本地模型数据，这些数据都存在着丢失的风险，当模型数据到达一定量级时，一些重复的模型也会占用本地存储资源。如果出现一款直接打开网站就能对模型数据进行管理并且预处理流程的算法执行速度与模型展示效果都能满足用户需求的话，那么这将会推进增材制造相关技术云服务化的进程。

鉴于以上几点原因，基于Web的增材制造预处理平台主要完成的任务是让用户可以上传下载并管理需要处理的模型，也可以在平台中选择其他用户的模型进行使用，同时还可以在平台上对模型进行基础的三维展示，拓扑重建，分层切片，轨迹规划以及GCode的生成。该平台适用于有3D打印需求的用户，实验室的科研人员以及普及3D打印技术的教育行业。

在确定需求时，我们主要通过与3D打印客户面谈，以及与相关技术从业人员进行讨论的方式来确定。结合多方观点和现实需求，我们会对整体系统的应用场景，系统模块，细分功能进行设计，以流程图和功能模块图的形式展现，最后与技术人员确定不同功能需要选择的数据结构和相关执行技术，总体上采用原型法的方式进行设计，并使用敏捷开发的方式去推进整个项目。

因此，基于Web的增材制造预处理平台的主要目标为：

1. 用户数据管理。
2. 模型数据管理。
3. 模型展示以及交互动画。
4. 实现增材制造预处理全流程。
5. 模型处理过程中的日志记录。
6. 实时看到当前页面性能。

## 涉及关键技术

### 增材制造预处理相关算法

#### STL模型

立体光造型文件（Stereo Lithographic File，简称STL文件）原本用于立体光刻电脑辅助设计，基本思想是利用离散的微小三角面片来组成各种三维模型的曲面。STL文件格式简单，容易输出，因此许多计算机辅助设计软件都可以输出STL格式文件[31]。STL格式文件被广泛应用与快速成型、3D打印和电脑辅助制造等领域。每个STL文件都是由一组笛卡尔坐标系下无序无规则的三角形顶点和法向量组成的，STL坐标必须是正数，不包含尺度信息，并且计量单位是任意的。如图 2‑1所示为一个STL格式文件的球体。



图 2‑1 STL模型

STL文件格式分为两种：一种是ASCII格式，另一种是二进制格式。具体的文件结构如下：

1. ASCII格式：

solid StlFileName // 文件路径及文件名

facet normal nx ny nz // 三角面片法向量的3个分量值

outer loop

vertex x y z // 三角面片第一个顶点坐标

vertex x y z // 三角面片第二个顶点坐标

vertex x y z // 三角面片第三个顶点坐标

end loop

end facet // 完成一个三角面片定义

......// 处理其他三角面片

endsolid StlFileName // 整个STL文件定义结束

1. 二进制格式：

UINT8[80] Header // 使用80个字节存储文件名

UINT32 Number of triangles // 模型的三角面片数量

// 每个三角面片使用50字节存储数据，具体规则如下：

REAL32[3] Normal vector // 每4个字节浮点数存储法线矢量的一个分量坐标

REAL32[3] Vertex1 // 12个字节存储1个顶点坐标

REAL32[3] Vertex2

REAL32[3] Vertex3

UINT16 Attribute byte count end // 2个字节存储三角面片属性

通过对STL文件存储格式的分析，本文以STL文件作为3D打印预处理平台的标准模型文件，并采用WebStorm作为开发平台，利用Three.js中Loader库的STLLoader类进行文件的读取加载，最后使用Three.js提供的api对模型进行绘制和显示。通过对源码的解读，STLLoader的读取流程如下：首先执行load函数通过一个URL访问到服务器端的模型资源，然后将模型文件通过ArrayBuffer的数据流接收到浏览器，之后通过parse函数对数据流进行解析，最后将解析结果交给WebGL内置引擎处理展示。模型读取效果如图 2‑2所示：



图 2‑2 STL模型读取实现效果图

#### STL模型拓扑重建与冗余去除

在上文叙述的STL模型文件结构中，每个面片结构都包含的各个点的信息，而通常情况下每个顶点都会被多个三角面片所共享，这里面就会导致数据的冗余，将冗余数据进行下一步处理，必然会导致重复计算，浪费内存空间增加计算时间。因此为了解决这个问题，我们需要对原始数据进行拓扑重建以及数据冗余的去除。

下面介绍顶点冗余去除方法[32]：对原始三角面片数组进行循环遍历，在遍历过程中，对每个三角面片顶点进行遍历，每读入一个顶点，首先判断该顶点是否在顶点数据结构中已经存在，如果已经存在，则存储该顶点的索引值到三角面片的数据结构中，否则表示该点为新点，存储进顶点表中。在判断顶点的过程中，我们会发现有些三角面片中点的距离其实很小，也就是说由于精度问题，过于小的三角面片其实是可以去除的。这里我们规定如果两个点，也就是一条线段长度小于一个极小值Min，比如本文设置的阈值是10-5，则就可以将这两个点当做同一个点来处理。其数学判别式如式 2.1所示：

（式 2.1）

其中p1、p2为两个顶点，x,y,z为其在笛卡尔坐标轴上对应的坐标。

在模型拓扑重构过程[33]中，我们需要对三个数据结构进行重建，分别是三角面片(Faces)、三角边(Edges)和顶点(Points)。为了方便后续切片分层以及路径规划处理的需求，数据结构应该满足以下几点特性：

1. 点存储结构记录了点坐标信息以及点索引。
2. 边存储结构记录了边索引、构成边的点索引和每条边的临接面索引信息。
3. 面存储结构包含了组成面的点索引，组成面的边索引以及面本身的访问索引。

具体数据结构定义如图 2‑3所示：



图 2‑3 拓扑重建数据结构

因为我们需要对这些数据进行频繁的查询操作，所以基于散列表的Map结构，也就是键值对集合是最适合用来存放重构后的数据。其中键值对构建映射关系的方法是哈希函数，也可以把这个Map结构看做一个哈希表[34]，哈希函数的选取会直接影响到程序的访问效率。在STL文件中，顶点数据通常都是Float类型，整数部分的位数往往没有小数位数多，且随着模型的增大，整数部分相同的数据也越来越多，因此我们选择的哈希函数要适当地对模型进行放缩处理，避免哈希冲突的情况产生。本文选择的哈希函数如下所示：

式中、、为顶点对应笛卡尔坐标系的坐标分量，为哈希表的长度，通过将顶点坐标放大整数部分，来防止因为整数部分相同导致的哈希地址冲突问题。

明确完数据结构，接下里就是详细的拓扑重建算法过程，如表 2‑1所示：

|  |
| --- |
| 拓扑重建算法流程 |
| 1：声明点、边、面数据结构并初始化；  2：按顺序读入三角面片信息；  3：将三角面片的三个顶点和三边计算哈希值，边的哈希值用临时变量存储起来，点数据则存入点哈希Map中；  4：从临时存储的三边信息中取出一组计算得出每条边的长度，如果小于设定的阈值，则继续4，大于设定的阈值则将边信息存入边哈希Map中，如果边信息读取完毕则进入5，反之继续执行4；  5：如果三边信息都存入了边信息Map中，则将三个顶点的索引和三边索引存入面片数组的点索引与边索引中，并且把当前面片信息存储到边数据结构中的面索引中；  6：判断三角面片读取完成是否为真，为假时返回3；  7：完成拓扑重建。 |

表 2‑1 拓扑重建算法流程

#### STL模型切片算法

STL模型切片就是根据起始高度、终止高度、切片层厚等参数来对网格模型进行平面截取的过程。而在每层处理的时候是通过截平面与处于该高度下的三角面片求交点，再将所求交点顺次连接成一个封闭多边形轮廓。STL模型分层切片示意图如图 2‑4所示：



图 2‑4 STL模型分层切片示意图

在已知文件的拓扑结构信息之后，我们做切片去寻找同一高度的点集时，就不用再将点进行排序，而是直接通过拓扑信息的邻接关系顺次将点连线，在三角面片搜索完成的同时也完成了对层的切片轮廓。

本文采用的是等厚分层切片算法，切片前需要知道分层方向、起始分层高度、终止分层高度和层厚这几个参数，从起始高度开始进行切片，先判断高度是否与模型会有交点，如果有交点则进行邻接三角面片搜索，得到该高度下的切片轮廓，再进行下一个高度的搜索直到搜索区间内的高度全部被搜索完毕。基于STL等厚分层切片算法的处理流程如图 2‑5所示：



图 2‑5 STL模型等厚分层切片算法流程

在上述算法流程中，找到与切平面有交点的边，求出与这条边的交点后通过提前存储的边结构中的邻接面去找到下一条与切平面相交的点，从而形成切平面与三角面片的交线，以此类推，就可以得到当前高度的封闭二维轮廓线数据。具体算法如表 2‑2所示。

|  |
| --- |
| 基于拓扑信息的切片算法流程 |
| 1：声明一个轮廓线存储数组result，并将提前拓扑构建完成的面数组中的每个数据打上hasSearch的标记，并初始化为FALSE，表示还没有遍历过；  2：从初始高度H开始，任意寻找一个未被搜索过的且包含当前切片高度范围的三角面片，如果有则进入3，并记录起始面片索引startIndex，没有则进入步骤7；  3：对该面片的三条边求交点，如果边与切平面共线则跳过该边，继续对其他边进行求交点；  4：如果总共得到一个交点则将该三角面片的hasSearch设置为TRUE，如果有两个不同的交点，则将其存放进一个临时数组temp[2]中，数组分别是两个点的信息，同时将数组push进result数组中，然后将该三角面片的hasSearch设置为TRUE；  5：将4中temp数组的最后一个点所在的边中的邻接面找到，并进入步骤3，直至该邻接面的索引为startIndex；  6：将高度H加上层厚t，继续2，直到H大于终止高度；  7：完成对文件的切片。 |

表 2‑2 基于拓扑信息的切片算法

#### 轨迹规划算法

等厚分层切片轨迹轮廓完成后，我们将要对封闭的轮廓线区域进行填充路径生成，通过用户设置的线宽，填充密度等参数去进行合理的路径规划。路径主要分为轮廓路径和填充路径两种，比较直观的区别就是轮廓路径填充打印机的每次路径的起点就是终点，因为轮廓路径填充只是将轮廓进行了等距向内或向外的偏置，不断沿着偏置的轨迹打印，直至填充满轮廓内部区域；而路径填充算法则是将特殊的线段按照一定的规律对内部区域进行扫描，所以路径可以为任意位置，每次打印完该线段后的位置跟下一个线段的起始位置都可能不同。

本文选择的是基于填充路径的扫描线多边形填充算法，基本原理是用水平线对轮廓进行扫描求交（由上到下），然后将交点进行排序输出，即完成填充工作。具体示意图如图 2‑6所示：

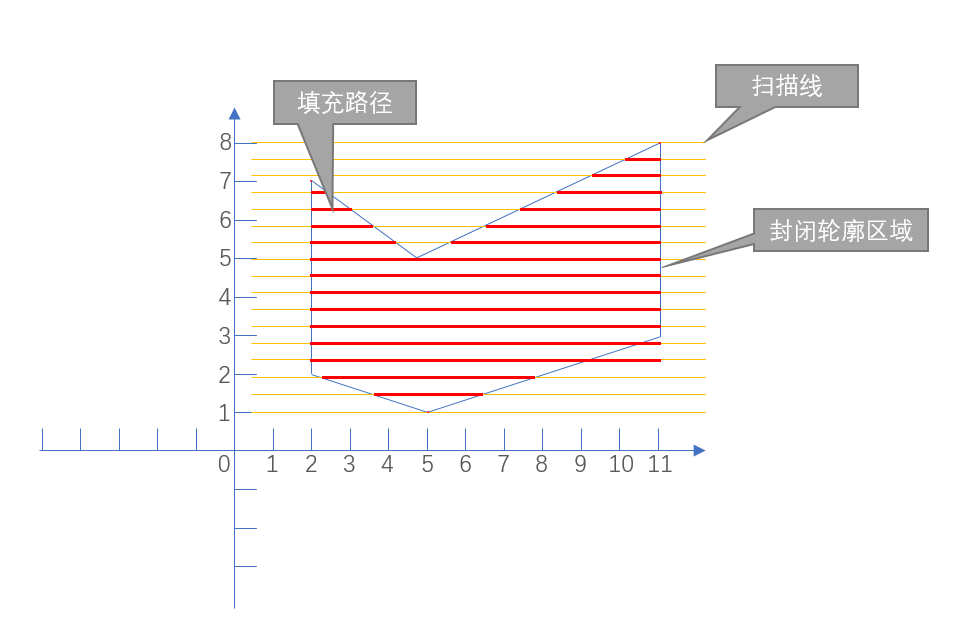


图 2‑6 扫描线多边形填充示意图

因为对于复杂轮廓或者模型中间有孔洞的三维模型，切片截面内部存在非打印区域，会形成内外轮廓[35]。在执行扫描线多边形填充算法前还需要对复杂轮廓进行内外轮廓判定，确定出需要被填充的区域。多轮廓图形的区域几何关系通常是包含与分离，我们区分内外轮廓的方法一般为：首先计算出每个轮廓四个方向的最值，即xMin、xMax、yMin、yMax，再根据这四个值的关系得出轮廓之间的位置关系，给每个轮廓初始化一个标记值等于0，再两两进行对比，如果一个轮廓被另一个轮廓包含，则给被包含轮廓的标记值加1，也就是记录该轮廓被包含的次数。由几何关系我们可以得知，当轮廓的标记值为奇数的时候，该轮廓为内轮廓；如果标记值为偶数，那么该轮廓为外轮廓。根据这个算法，我们就可以很快的得出每个轮廓的内外轮廓关系，只有内外轮廓间的区域需要进行填充路径规划。

根据上述算法我们可以划分出多轮廓的填充区域，对于一些复杂模型的内外轮廓区分如图 2‑7所示，图中的绿色线条表示外轮廓线，红色线条表示内轮廓线，蓝色线条表示需要轨迹填充的区域。



图 2‑7 复杂模型内外轮廓示意图

#### GCode生成算法

打印路径确定后，我们数据是按照Three.js进行绘制的数据格式存储的，而要让打印机执行打印指令，还需要将计算出来的路径转化为3D打印机可识别的电路代码，而不同的打印机有着不同型号的运动控制器，因此必须转化为一种都遵循的标准代码，也就是GCode。GCode原本是用来控制数控机床的加工模型的一种语言，比如控制机床的切削工具按照指定的路径对零件进行切削剪裁，形成所需的零件[36-38]。由于GCode的结构清晰，规则完善简易，而且代码的可读性也很高，用户可以很方便的观察到加工路径的位置坐标，所以我们使用GCode作为3D打印运动控制器的控制代码。

GCode代码的格式由一个英文字母（A-Z）加一串数字（0-9）组成，字母G开头的指令是用来控制打印机喷头运动，字母M开头的指令是一些辅助命令，字母X，Y开头的指令是用来控制喷头在X轴、Y轴上运动位置的增减，字母E开头的命令代表着喷头出丝量，字母F开头的指令表示喷头移速。常用G代码指令含义如表 2‑3所示：

|  |  |
| --- | --- |
| 字段 | 含义 |
| Gddd | 控制喷头运动 |
| Mddd | 辅助命令 |
| Tddd | 选择哪种打印头 |
| Sddd | 是否检查限位开关 |
| Pddd | 命令参数，例如时间 |
| Xddd | 坐标点的X轴位置 |
| Yddd | 坐标点的Y轴位置 |
| Zddd | 坐标点的Z轴位置 |
| Eddd | 喷头出丝的挤出量 |
| Fddd | 打印喷头移动速度 |
| Rddd | 温度相关参数 |
| Qddd | 目前未使用参数 |
| Nddd | 行号 |
| \*ddd | 校验码，检查通讯错误 |
| Iddd | 目前未使用参数 |
| Jddd | 目前未使用参数 |

表 2‑3 常用G代码指令含义

在打印机读取GCode命令前，我们需要对打印机进行初始化：喷头位置归零、对打印材料进行加热融化、加热底部平台、抬起打印喷头、喷头出丝、打开制冷风扇冷却路径。每层打印开始前，我们要将喷头的位置移动至该层高度的Z坐标处，根据路径数据，每个点都是一行GCode指令，指令里包含了喷头移动速度、喷出材料的长度和宽度、坐标点的位置等，喷头的行进速度是可以根据需要调整的，空行程的代码为G0，否则为G1。当前层打印结束后，关闭加热装置，打印喷头回到原点。

GCode代码中我们需要关注的就是控制运动的G代码和M代码，为了实现路径向G代码的转换，我们需要对部分G代码和M代码进行了解，常用代码定义如表 2‑4所示：

|  |  |
| --- | --- |
| 代码 | 含义 |
| G0 | 快速直线运动 |
| G1 | 控制直线运动 |
| G2 | 顺时针圆弧运动 |
| G3 | 逆时针圆弧运动 |
| G4 | 暂停运动 |
| G28 | 归零 |
| G91 | 定义当前位置 |
| M0 | 停止 |
| M1 | 睡眠 |
| M23 | 选择GCode文件 |
| M33 | 获取GCode文件完整路径 |
| M104 | 设置打印机喷头温度 |
| M114 | 输出当前位置 |
| M251 | 保存当前Z轴高度 |

表 2‑4 GCode常用代码含义

### Web应用开发相关技术

Web应用开发技术栈经过这几年的发展，可以说是百家争鸣，在基于前后端分离的模式下，主流前端框架有Angular、React和Vue这三种，服务端主要有Java、PHP、Golang、Python和NodeJS，数据库的选择就更多了，例如MySQL、Oracle、Redis、MongoDB等。鉴于我们系统的计算模块使用的都是本地用户的计算资源，并且模型数据的结构是JSON格式的文档流数据，所以我们选择的是一个轻量型架构，并且为了方便代码编写，减少学习成本，我们选择编程语言统一的框架，因此我们是基于VUE+Ant Design Vue+Three.js+Express.js+MongoDB来对系统进行开发。

#### VUE框架

Vue[39]是一套用于构建用户界面的渐进式框架，Vue设计的核心是让用户只关心视图层，相较与其他大型框架，不仅易于上手，还便于与第三方库进行整合。Vue最大的特点有四个：MVVM设计模式、虚拟DOM、组件化开发、服务器单独部署。

1. 前端MVVM

MVVM模式是从MVC设计模式[40]演变而成，MVC设计模式如图 2‑8所示,分为Model（数据模型层）、View（视图层）、Controller（控制层）。在前后端未分离时，开发是基于JSP（JavaServer Pages），其处理流程为：①客户端向服务器发起HTTP请求。②服务端根据请求的URL去执行内部的业务逻辑。③服务端将业务代码执行的结果和静态页面相结合生成HTML模板并返回给客户端。④客户端浏览器解析HTML文件，并将数据渲染到网页上。服务器遵循MVC的设计模式，其中M表示服务器中设计好的输入输出数据结构，C表示接收和发出各类业务数据，V表示将数据结构渲染为HTML模板。而VUE所依赖的MVVM模式[41]如图 2‑9所示，MVVM与MVC最大的区别就是：它实现了View和Model的自动同步，也就是当Model的属性改变时，我们不用再自己手动操作Dom元素，来改变View的显示，而是改变属性后该属性对应View层会自动改变。这极大地方便了开发者对DOM的操作，提升了开发者的工作效率。



图 2‑8 MVC模式



图 2‑9 MVVM模式

1. 虚拟DOM

Vue视图渲染是过程使用的是虚拟DOM的技术，浏览器渲染引擎工作流程大致分为5步：创建DOM树，创建StyleRules，创建Render树，布局Layout，绘制Painting。我们传统的开发模式，当页面的DOM节点变动后，浏览器会从创建DOM树开始从头渲染，因此每个微小的变化都会导致页面重刷新，因此计算代价就十分昂贵。而虚拟DOM技术则是将页面的多次变化内容保存到本地的JS对象当中，再根据diff算法去构建新的DOM树，最终将这个对象一次性挂载到真实DOM树上，避免了大量的重复计算，提升用户体验。

1. 组件化开发

Vue的组件化开发，支持将小型、独立和通常可复用的组件拿来构建大型应用，对组件成员都提供了相应的管理机制，方便开发者进行注册或实例化，比如通常一个应用会以一颗嵌套的组件树的形式来组织，如图 2‑10所示：



图 2‑10 应用的组件化形式

1. 服务器单独部署

在应用开发阶段，Vue项目是在Node.js的环境下进行开发的，使用了NPM、Webpack为工具来部署Web服务，辅助开发人员对代码进行调试，例如：①Vue应用在开发中，代码做了任何修改都能实时显示到Web服务中，并且基于NodeJS的服务具备反向代理功能，能够将网页向后台发送的AJAX请求转发到后端服务器。②基于NodeJS的后端Mock也能模拟后端响应HTTP请求，返回测试JSON数据，已达到模拟规范数据进行独立开发的目的。

而实际生产阶段，则需要将项目代码编译打包成静态的HTML、JavaScript以及CSS文件，然后部署到Nginx或者Apache服务器上完成前端应用部署。Vue前端开发体系，因为是基于前后端分离模式，所以无论后端采用何种技术框架，只要双方提供约定好的Restful风格的API即可完成数据的交互。

#### Ant Design of Vue

Ant Design of Vue是基于Ant Design设计体系的Vue UI组件库，开发和服务于企业级产品。由阿里巴巴公司的蚂蚁金服团队开发并开源，包含了Web应用常见的组件库，主要有以下三大特点：一是提炼自企业级中后台产品的交互语言和视觉风格，具有较为良好的用户体验；二是开箱即用的高质量Vue组件，省去开发者大量重复造轮子的时间；三是共享Ant Design of React设计工具体系，降低开发者学习成本。

#### Three.js

在Web端进行三维绘图，肯定离不开WebGL技术，该技术标准基于JavaScript和OpenGL ES2.0，它的特点就是不需要配置额外的绘图渲染插件比如Flash，只需要浏览器环境就能操作复杂的三维结构和3D游戏。一般网页主要是使用HTML、JavaScript以及CSS进行一些动态效果的呈现，比如使用CSS3绘制一些页面元素的动态过渡效果。而WebGL则是一种更为复杂的网页展示技术，借助了系统显卡来对图形数据进行加速处理[42]。

本文使用到的Three.js是一个Github的开源项目，是目前在Web应用用来处理3D程序的技术中生态最为庞大的项目。Web前端开发人员如果直接使用WebGL规范提供的API进行编程，就需要开发人员掌握很多数学知识、图形学知识才能完成3D编程任务，并且基础操作的编码量巨大，而Three.js则是对这些基础API做了二次封装，可以让用户按照一定的规律去绘制各种复杂的图形，并且提供很多对象交互接口和多种动画生成方法。大大地降低了三维模型处理程序上手开发的难度。

三维图形处理跟二维平面图像元素处理方式差别较大，因此在使用Three.js前需要掌握一些3D编程的基本概念，主要包括以下几个基本要素：场景、相机、渲染器。Three.js程序结构如图 2‑11所示，其中场景就是一个虚拟三维空间，承载所有物体的容器；相机则是控制视线的位置、方向，角度，相机内的画面就是我们在编程区域所能看到的内容；渲染器就是代表着渲染的结果应该绘制在页面的哪个元素上，并且用什么方式来进行绘制。



图 2‑11 Three.js程序结构

#### Express.js

Express.js是一个基于Node.js平台的快速、开放、极简的Web服务开发框架[44]。Node.js是在2009年由Ryan Dahl开发的，最初始于他的一个项目，该项目集成了Google V8 JavaScript引擎、事件循环和底层I/O应用编程接口的平台[43]。Node.js具有一下三大特点：①单线程，NodeJS服务不会为每个连接创建一个新的线程，而是只使用一个线程，当有用户连接，就触发一个内部事件，通过非阻塞I/O、事件驱动机制在宏观上模拟并行机制。②非阻塞I/O，比如我们在访问数据库时，传统的单线程处理机制会在执行访问数据库代码后暂停整个线程，直到数据库返回结果，才能继续执行后续代码。而非阻塞I/O机制则是将数据库返回结果后的处理代码放在回调函数中，保证主线程的代码继续执行。③事件驱动，NodeJS中当用户建立请求连接，提交数据等操作时，会触发相应事件，而在每一个时刻，只能执行一个事件的回调函数，但是在执行事件回调的过程中，还可以转去处理其他事件，然后返回继续执行原事件的回调函数，这种处理机制，也被叫做“事件环”机制，事件驱动模式如图 2‑12所示。



图 2‑12 事件驱动模式

前文已经介绍了Node.js，而在本项目中我们使用Express.js去创建一个可维护的Web服务端应用程序，它提供了强大的API，可以有效的处理路由、HTTP请求、静态文件等，相当于是一个轻量级的Java后台的作用，因为使用的是JavaScript进行编写，所以对于开发者而言，也降低了上手难度，不用去学习多种语言，就可以搭建一套属于自己的后端请求接口服务系统。

#### MongoDB

随着Web2.0的兴起，非关系型数据库[45]成了热门的研究领域。因为传统的关系型数据库在面对大规模用户、海量数据以及高并发的SNS（Social Networking Service）实时动态网站时，会产生许多问题：①面对高并发需求会出现性能瓶颈[46]，比如微博这种大型SNS网站，同时在线用户很可能会达到百万，那么数据库就会出现高并发负载，经常会达到上美秒万次请求。传统关系型数据库在面对每秒上万次查询还能承受，但是面对上万次数据库的读写请求时，磁盘的I/O将无法支撑。②对于海量数据无法高效存储和访问，比如一个用户数据里面订阅了多个栏目，每个栏目里面又包含了不同种类的刊物，每个刊物里面又关联了作者，此时需要查阅该用户下有多少个服务的作者，关系型数据库通常会分库分表切分数据，但是一旦遇到高并发请求，查询效率会变得特别低下，并且增加了后期数据库维护和迁移的难度。③关系型数据库具有ACID[47]特性，可是大部分网站中，很多数据操作都无法严格满足事务的特性，因此高负载下的数据一致性难以保证。

而非关系型数据库（Nosql）慢慢成为主流，从本质上两者区别在于数据结构，非关系型数据库通常使用的数据结构是键值式和文档式。都有着以下几个特点：非关系型、分布式、高扩展性、读写性能高。本文中因为要对STL模型数据进行存储，而STL模型被导入到Web端时的数据是JSON格式，而且还要与用户进行关联存储，所以文档式的数据库最符合系统需求。MongoDB是最有代表性的文档式非关系型数据库，所以我们选择MongoDB作为本系统的数据库。

MongoDB是一个面向集合（Collection）、模式自由（Schema-Less）、文档型（Document）的数据库。面向集合指的是数据被划分存储在数据集合里面，有点类似关系型数据库的表概念；文档型指的是数据存储在文档中，每个集合可以包含无数个文档，文档本身就是一组键值对，键值对的键可以是任意数据类型，这种格式也被叫做Bson，类似于Json的二进制数据格式，两者都支持内嵌对象和数组，但是Bson有着更加丰富的数据类型；模式自由指的是集合中的文档不一定要遵守相同的格式，只要是数据就都能往里面存储，不用担心数据结构不一致而去分集合。选用MongoDB数据库还有一个最大的优点，它是基于磁盘的数据库，面临海量数据的读写时仍然有高效率的I/O。官方测试结果表明，当数据量大于50GB时，MongoDB要比MySQL快将近10倍。

## 本章小结

本章主要介绍了基于Web的增材制造预处理平台所要用到的关键技术及原理，根据3D打印的过程，介绍了增材制造预处理过程中涉及到的数据结构，详细阐述了STL模型拓扑重建算法，切片算法，轨迹规划算法和GCode生成算法的原理；然后分析了该Web系统所需要的技术栈，从前端Vue框架、前端UI库Ant Design of Vue，到Web3D开发框架Three.js，然后后台服务端框架Express.js以及数据库MongoDB的对比选型。下一章将进行系统需求分析和概要设计。

# 系统需求分析及概要设计

## 需求分析

增材制造数据预处理平台需要给用户提供模型数据的展示和模型处理功能，能让用户在系统里进行模型分层切片，打印轨迹规划，打印过程动画模拟以及记录模型处理日志等操作，此外本系统作为3D打印预处理过程云服务化的探索，我们还需要关注到用户本身的数据管理问题，用户可以登录注册，并且拥有个人的在线模型仓库，这样用户可以将模型数据存放在云服务器上，随时修改模型相关信息或者下载到本地使用，也可以共享给其他用户；此外还需要系统管理员来对所有的用户信息和模型信息进行管理。

本文对3D模型的渲染绘制主要使用Three.js技术来构建画面场景，使用Three.js提供的API绘制算法处理后的图形数据以及对模型进行缩放、平移、旋转等交互操作；针对涉及到的处理算法我们使用JavaScript内置的数据类型来对算法进行复现；前端页面使用Vue框架来进行开发；页面路由控制使用Vue-Router进行配置；前端不同页面之间的共享数据使用Vuex来进行管理；前后端通信的HTTP请求使用Axios进行处理；后端使用Express.js响应接口并同步操作数据库；数据则使用MongoDB数据库来进行存储。

## 系统用户及用例

用例图（Use Case Diagram）描述了用户如何去使用这个系统，是软件需求分析中十分重要的步骤[48,49]。基于Web的增材制造预处理平台主要分为模型预处理平台和后台管理两个子系统，系统用户主要分为普通用户和管理员两类。普通用户在系统里可以完成对模型数据的预处理流程，管理员是负责对整个系统的用户信息和模型信息进行维护。下面对系统的用户进行用例分析，并对每个用例进行详细说明。

1. 系统管理员的用例分析

系统管理员用例图如图 3‑1所示。

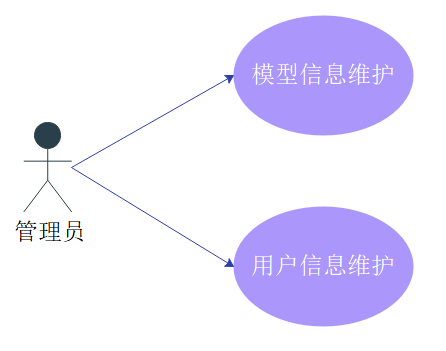


图 3‑1 管理员用例图

系统管理员主要负责模型信息和用户信息的维护，模型信息管理就是可以对模型增加、删除、修改模型附属的一些信息；用户信息管理就是可以增加、删除、修改用户的基本数据。

1. 普通用户的用例分析

普通用户用例图如图 3‑2所示。



图 3‑2 普通用户用例图

针对普通用户的用例进行分析：①首先将本地的模型数据上传到个人仓库中，并且可以修改模型相关信息，也可以把模型放入公共仓库中。②用户可以在系统里预览模型数据，并提供放缩、旋转、还原和显/隐图层等交互操作。③用户可以对模型数据使用切片分层，轨迹规划，轨迹动画以及生成GCode等功能。④用户可以在日志列表里查看每次处理的一些信息，并可以导出记录。

## 系统业务流程分析

基于Web的增材制造预处理系统的核心流程如下：

1. 注册系统用户，注册完成后登陆系统进入模型管理中心。
2. 用户上传本地模型到个人模型仓库内，并设置模型相关的参数，也可以直接从所有模型库里选择需要的模型放入我的模型库里。
3. 选择一个模型进入工作台，在工作台页面可以对模型进行放缩、回中、旋转、显隐坐标轴、显隐底部参考网格、显隐模型等展示操作。
4. 对模型进行切片处理，并将切片结果渲染到画面上，更新图层数据，同时加入日志记录。
5. 再对模型进行轨迹规划处理，渲染画面，更新图层数据，并把记录加入日志。
6. 同时还能按照轨迹规划的路径生成相应的轨迹打印动画。
7. 最后生成3D打印机能识别的GCode，可以将GCode文件下载到本地。
8. 所有的模型处理操作都记录在了日志面板里。

具体流程如图 3‑3所示：



图 3‑3 系统主要使用流程示意图

## 系统详细功能模块设计

### 注册登录模块

注册模块流程是在注册页面输入用户名、真实姓名、密码三个数据并点击提交按钮，将数据交付于服务器，服务器先对用户名进行查询，如果有重名用户则返回注册失败，否则将数据插入到用户集合当中，并返回注册成功，具体注册流程如图 3‑4所示：



图 3‑4 注册流程图

登录模块使用了JWT（Json web token）进行校验，具体流程就是在每次登录的时候服务器会根据规范生成一个令牌（Token），本文选择的令牌生成算法是HS256，并且把该令牌放在请求头部返回给客户端，此时用户在之后的请求中都把该令牌带上，这样就可以保证服务器能识别出各个客户端用户的身份。具体验证流程如图 3‑5所示：



图 3‑5 登录验证流程

### 用户模型管理模块

用户模型管理模块分为我的模型库和所有模型库，在我的模型库里面，用户可以上传本地模型文件到我的模型库，在上传的同时需要输入模型名称、模型描述、模型权限、模型图片、模型文件这几个数据，模型上传成功后用户还能对模型数据进行修改和删除操作，在修改模型信息页面还支持模型图片和模型数据的下载功能；所有模型库展示了系统所有用户上传的模型文件，对于公开权限的模型，用户可以直接将模型加入我的模型库，而私有的模型则需要向该用户提出使用申请，如果同意则会自动加入到我的模型库中。模型库管理模块具体功能如图 3‑6所示：



图 3‑6 模型管理模块功能示意图

### 模型显示交互模块

模型展示以及交互操作我们使用的是Three.js这个开源库，它帮助开发者将很多WebGL底层的API进行了二次封装，避免将常规操作进行多次繁琐的组合封装。模型展示与交互部分主要分为基础显示、视窗放缩、视窗回中、图层显/隐控制、算法处理结果、页面性能面板、动画演示这几部分组成，其中图层显/隐控制包含的对象有包络盒、目标模型、底部网格、坐标轴、水平切片、切片轨迹；而算法处理结果则包含了轮廓线信息以及轮廓填充轨迹信息。具体模块关系如图 3‑7所示：



图 3‑7 模型展示交互模块示意图

### 增材制造预处理模块

在第二章我们分析了增材制造预处理过程所需要的几个步骤，首先导入STL模型文件，再对导入的模型进行冗余去除和拓扑重建，再使用重建后的模型数据进行切片生成，然后对每个切片的轮廓进行轨迹填充，最后将轨迹代码转化为打印机可执行的GCode。增材制造预处理的模块设计如图 3‑8所示：



图 3‑8 增材制造预处理流程模块示意图

### 系统用户信息管理模块

系统用户信息管理模块是提供给管理员使用的功能，包括用户信息搜索，用户信息查看，用户信息修改，用户信息删除。并且搜索功能可以根据用户名与真实姓名对用户数据库进行联合模糊查询，方便管理员对系统用户进行管理，详细结构如图 3‑9所示。



图 3‑9 用户信息管理模块

### 系统模型库管理模块

系统模型库管理模块是可以让管理员对平台中所有模型进行管理，提供了模型查询，模型数据修改，模型删除功能。模型搜索关键词可以是所属用户的用户名、模型名称或模型描述，具体结构如图 3‑10所示。



图 3‑10 系统模型库管理模块

## 本章小结

本章首先介绍了基于Web 的增材制造预处理平台的软件需求，然后根据需求分析将系统用户分为普通用户和管理员两种，并对每个角色进行了详细的用例建模。之后对平台的核心业务流程进行设计概述，然后再对系统进行子功能模块设计，并结合Web开发的特点对每个模块的内部流程进行详细设计。下一章我们将叙述整个系统的实现过程与实验测试结果分析。

# 增材制造预处理Web平台实现与测试

## 开发环境搭建

### 搭建Node.js服务器

因为JavaScript的代码需要在Node.js的服务端上运行，所以我们首先进行Node.js的安装，在官网上（https://nodejs.org/zh-cn）选择对应系统的安装包，根据提示完成Node.js服务的安装，因为现在版本的Node.js默认包含了包管理工具Npm，因此我们可以通过命令行输入node –v和npm –v去测试Node.js和npm是否安装成功，安装成功后会出现版本的信息，如图 4‑1所示。



图 4‑1 Node与npm安装成功示意图

### 数据库安装

进入MongoDB官网找到对应系统安装包，下载完成后需要配置数据存储位置，数据分为数据库数据和日志数据，本文将数据存放在C:\MongoDBData路径下。接下来需要执行mongod.exe文件来安装MongoDB服务，我们在安装路径中的bin目录下打开命令行，输入C:\mongodb\bin\mongod.exe --config "C:\mongodb\mongod.cfg" –install来安装MongoDB服务。最后我们通过net start MongoDB和net stop MongoDB来启动和关闭MongoDB服务，效果如所示。



图 4‑2 MongoDB数据库服务启动与停止

### 代码编辑器选择

本文中涉及到的编程语言为JavaScript，因为JavaScript是一门解释性脚本语言，不需要提前编译，浏览器就能直接执行，所以记事本编写也能被运行，但是为了快捷方便地进行编码，我们需要代码提示、打点调试、颜色标记等辅助功能，因此需要选择一个合适的代码编辑器。本文使用的是WebStrom编辑器，这是JetBrians公司旗下的一款JavaScript开发工具，支持多种语言框架，内部涵盖大部分功能插件，且对许多工具进行了集成，如npm、git、yeoman等。软件安装只需在官网（https://www.jetbrains.com/webstorm/）下载对应系统的版本进行安装即可。

### 软硬件平台

|  |  |  |
| --- | --- | --- |
| 序号 | 名称 | 配置 |
| 1 | 处理器 | Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz |
| 2 | 机带RAM | 8.00GB |
| 3 | 显卡 | NVIDIA GeForce GTX 1050 Ti |

表 4‑1 开发硬件配置

|  |  |  |
| --- | --- | --- |
| 序号 | 软件 | 说明 |
| 1 | Windows 10 | 操作系统 |
| 2 | WebStrom | 代码编辑器 |
| 3 | Node.js | JavaScript运行环境 |
| 4 | Chrome浏览器 | Web应用运行环境 |
| 5 | Three.js | 图形开发包 |
| 6 | Vue | 前端开发包 |
| 7 | Express.js | 后台开发包 |
| 8 | Mongoose | 数据库开发包 |

表 4‑2 开发软件配置

## 前端实现

### 前端项目初始化

本文前端开发使用到的框架是Vue，对于Vue项目的初始化我们可以使用官方提供的脚手架工具Vue-cli，它为开发者提供了一个很好的常规项目模块，避免开发者自行去配置一些打包转码等重复性处理操作。首先打开命令行使用官方命令npm install -g @vue/cli进行安装，然后在项目目录用命令行执行vue create web-three并选择相关配置参数，然后等待项目创建，程序执行完毕后我们会看到目录下多了一个

web-three的文件夹，也就说明项目初始化完毕。进入该文件夹后在命令行执行npm run serve就可以启动项目，项目成功启动后如图 4‑3所示。



图 4‑3 Vue项目成功启动示意图

项目创建成功后我们使用WebStrom打开项目文件，对项目的目录树进行分析。其中node\_modules文件存放的是整个项目的开发依赖包，public目录下则是存放网站图标和入口Html文件；src文件夹下是项目的源码，其中assets文件夹下存放的是一些静态资源，而components文件夹则是Vue的业务组件代码，main.js是整个项目的入口文件，App.vue是页面实例化的入口文件；package.json文件定义了开发所需要的各种模块以及项目的配置信息，babel.config.js文件是ES6语法编译配置；最后README.md是Markdown格式的文件，里面包含了项目如何启动、打包等信息。项目初始化结构如图 4‑4所示。



图 4‑4 项目初始化目录结构

根据主流的前后端分离架构，前端需要负责页面展示，业务逻辑控制，数据输入校验，页面路由跳转，页内数据传递以及定义接口字段。这些功能都有不同的开发包来进行处理，为了规范编码流程以及合理地对代码模块进行管理，我们需要改造初始化的项目结构，并引入相关资源依赖包来支撑后续代码的编写。经改造后的项目结构如所示。我们在src目录下新增了api文件夹，用来管理应用中涉及到的各个Http请求；新增js文件夹，里面存放的是绘图操作以及模型处理过程中涉及到的算法实现代码；新增routers文件夹，该文件夹下存放着路由控制以及每个页面的详细实现代码；新增store文件夹，我们把页面之间需要传递或者共享的数据放在该文件夹内进行管理；新增vue.config.js文件，里面是项目进行构建打包，部署到服务器上需要进行的优化配置。经过重新改造后的目录基本涵盖了目前常规Web应用开发所能涉及到的模块。



图 4‑5 改造后的前端目录结构

接下来进行开发依赖包的选择，我们可以在package.json中看到该项目运用到了哪些公共模块包，一般分为运行依赖包和开发依赖包。运行依赖包如图 4‑6所示，开发依赖包如图 4‑7所示。



图 4‑6 项目运行依赖包



图 4‑7 项目开发依赖包

### 页面路由设计

路由在网页中对应的是不同的url下渲染的页面，在第三章里我们介绍了系统的核心流程，所以该系统所涉及到的页面主要分为登录页面、模型文件管理页面、工作台页面、管理员页面，因此我们的路由就需要有四个模块，分别展示对应的页面。我们在router文件夹的index文件里进行路由配置，首先设置路由的模式，在Vue-router中有两种模式，一种是hash模式，一种是history模式，两者最大的区别就在于hash模式的url地址是http://www.mysite.com/#/hello，而history模式的url地址是http://www.mysite.com/hello，前者在请求服务器资源时候不会带上#后面的参数，而后者则是请求完整的url路径地址，如果后端没有做相应路径的资源配置，就会出现404页面。由于我们是前后端分离的应用，前端自己有一个运行服务器对路由进行处理，所以刷新后丢失资源的问题已经得到很好的解决，并且history模式更符合url地址规范，去除#号后显得更加美观。根据上述页面的需求我们设置了如下几个路由：/login路由下的登录页面、/file路由下的文件管理页面、/work路由下的工作台页面和/admin路由下的管理员页面。详细配置如图 4‑8所示。



图 4‑8 项目路由详细配置

### HTTP请求封装

前后端分离的模式下，数据传输是通过RESTful风格的接口来实现的。HTTP请求方法一共有9种，分别是GET、POST、 HEAD、OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT 方法，本文中用到了GET和POST这两种方法。由于从HTTP层面上来说，客户端收到的状态码是有限的，而且针对不同的接口都有可能出现这些状态码，因此在调用接口时我们需要对请求接口做一次封装；并且由于业务因素，相同的接口很可能会在不同页面多次调用，所以还要根据后端数据类型进行统一分发调用，这样就可以让代码更精简且具有较高的可维护性。

下面来详细介绍HTTP请求的封装过程，首先在api文件夹里新建baseUrl.js和axiosService.js文件，前者配置了不同路由下所控制的资源，后者是对所有接口在通信层面上的处理。在baseUrl.js里面我们配置了后端服务的ip地址，根据后端不同的部署位置进行设置，distributeUrl中存放了后端服务根地址、用户数据控制地址和模型数据控制地址，如图 4‑9所示。在axiosService.js中我们首先创建了axios实例，



图 4‑9 baseUrl.js文件代码

设置了请求的过期时间，然后设置POST方法的请求头，在请求拦截器中将后端返回的Token值放入请求头部中，方便后端进行用户权限校验，然后在响应拦截器中处理不同的状态码，如果请求的返回状态码为200或者201直接将后端返回的数据resolve，状态码为401表示登录过期，将页面跳至登录页面重新登录，状态码为403表示接口权限不够，状态码500返回服务器内部错误，状态码502返回网络问题，具体处理函数如图 4‑10所示。



图 4‑10请求返回状态码统一处理

接口通信层面封装完毕后我们需要对每个数据的处理接口进行分发，我们在apiSugar里创建一index.js入口文件对所有数据控制接口进行分发，然后在modelController.js和tokensController.js中进行每个接口进行封装，以tokensController.js文件为例，首先导入接口域名列表和通信层封装后的axios接口函数，针对不同的请求方法进行export处理，每个函数中的params是接口需要给后端传递数据的形参，具体实现如图 4‑11所示。



图 4‑11 用户数据接口实现

### 系统页面实现

每个Web页面都是由HTML、CSS、JS三部分组成，本文使用的Vue框架在每个页面组件中的代码结构如图 4‑12所示。Style标签里放置当前组件的CSS代码，Template标签里编写页面HTML骨架代码，Script标签中是当前组件的逻辑控制代码。整个页面的样式组件是使用蚂蚁金服开源的UI框架Ant Design of Vue，下面是系统各个页面的实现：



图 4‑12 Vue页面基本结构

1. 登录注册页面：包含平台的功能说明，登录表单交互，注册表单交互。



图 4‑13 登录注册页面

1. 模型管理中心：我的模型库页面，包含了模型数据展示、模型数据修改表单、上传模型表单和删除提示；所有模型库页面，包含了模型数据展示、加入仓库和申请使用；个人中心页面，个人资料显示和个人资料修改。



图 4‑14 我的模型库页面



图 4‑15 所有模型库页面



图 4‑16 个人中心页面

1. 工作台页面：顶部菜单栏（分为文件、图层和功能，文件有另存为、退出编辑，图层有包络盒、目标模型、底部网格、坐标轴等显/隐控制，功能包括切片、轨迹生成、G代码、动画和日志）；左侧菜单栏（含有还原、放大、缩小操作）；右上方有JavaScript性能监听器（包含帧率、延时、内存监听）。



图 4‑17 工作台页面



图 4‑18 顶部菜单栏



图 4‑19 JavaScript性能监听器

1. 管理员页面：分为用户管理页面和模型管理页面，用户管理页面包含系统用户的增删改查，模型管理页面包含STL模型文件的查看和删除。



图 4‑20 管理员页面

## 模型交互实现

模型交互的实现需要理解Three.js中的三大组件：场景（Scene）、相机（Camera）和渲染器（Renderer）。只要是依赖Three.js进行Web端的3D应用开发，必须按照创建场景、设置相机角度、设置渲染器窗口参数这个流程才能将预计的场景渲染到网页上去。接下来我们会对系统中涉及到的每个模型处理模块进行分析。

### 场景初始化

进入工作台页面的时候整个三维场景是由灯光、相机、背景、坐标轴、底部网格组成，所以我们在页面渲染前需要对以上部件进行初始化。灯光初始化本文选择了两个处于Z轴的点光源，颜色为0xffffff，方向互为反向，此外还加入了一个颜色为0xffffff的环境光，具体实现代码如图 4‑21所示。



图 4‑21 灯光初始化代码

相机在三维场景中就类似现实世界中的人眼去观察世界，在Three.js中的相机对象分为正投影相机（OrthographicCamera）和透视投影相机（PerspectiveCamera），如图 4‑22所示我们可以看到正投影下，线段的角度不同投射到投影面上的长度就不同，而透视投影在投影面上的长度不仅与线段的角度有关，还与观察距离有关。因为本系



图 4‑22 正射投影和投射投影原理

统会涉及到对物体的放缩，如果使用投射投影的话，当距离太远会导致物体消失在页面视野中，而正投影是不会出现物体消失在视窗内的现象，所以我们选择正投影相机来作为三维场景的观察器，具体初始化代码如图 4‑23所示。

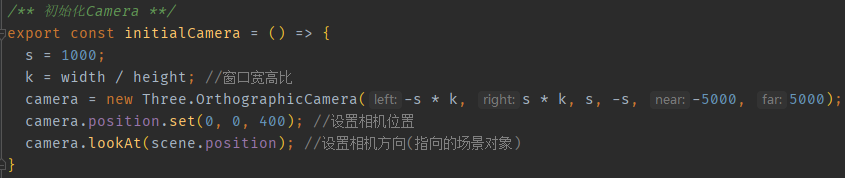


图 4‑23 相机初始化代码

背景指的是整个三维场景的背景画面，Three.js支持纯色和自定义贴图进行渲染，因为背景不是系统的主要展示区域，所以我们将画布背景设置为0x333333的纯色。

坐标轴的初始化是为了让模型导入后给用户一个可参考的坐标系，方便用户查看模型。具体绘制算法为：先定义dirX、dirY、dirZ三个分别指向X、Y、Z轴的向量，并将它们的长度规定化为1，然后再创建Three. ArrowHelper箭头对象，并设置三个方向的颜色和长度，再将定义好的三个向量传入箭头对象中，这样就可以实例化出三个指向XYZ轴的箭头，并将其加入Scene对象中。

底部网格也是用来给用户提供一个模型范围参考区域，具体绘制算法为：根据模型的尺寸设定网格矩形区域的长宽，然后在X轴和Y轴方向各定义两条Line对象，然后将这两个Line对象进行等距复制，直到满足长宽区域被横竖线段填充完毕，最后将若干条Line对象加入Scene对象中。

当需要初始化的对象都被添加进整个场景的Scene对象中后我们要开始执行渲染操作，我们先通过Three. WebGLRenderer来创建渲染器对象，再通过渲染器对象暴露出来的render函数执行渲染操作，最终初始化效果如图 4‑24所示。



图 4‑24 三维场景初始化效果

### 模型加载与导出

1. 模型加载：用户在我的模型库页面选择符合条件的模型数据进入工作台页面，然后前端会根据选择的模型访问地址去向服务器发送请求访问服务器上的文件资源，服务器接收到请求后将数据返回给前端，前端接收到服务器返回的二进制流数据，并同步使用Three.js提供的STLLoader处理函数将ArrayBuffer格式的数据解析成方便WebGL渲染的JSON格式数据，流程示意如图 4‑25所示，模型导入效果如图 4‑26所示。



图 4‑25 模型导入实现流程



图 4‑26 模型导入效果图

1. 模型数据导出：我们通过遍历Scene将页面上Visible属性为True的对象添加进新创建的Scene对象中，然后再把新的Scene对象数据转化为Blob二进制格式再使用STLExporter函数将其进行导出，效果如图 4‑27所示。



图 4‑27 模型导出效果示意图

### 场景基本交互操作

1. 还原：模型在鼠标操作过程中会被旋转到各个角度，而我们有时候希望一键将模型回正，这就需要将视窗角度重新调整到初始位置。具体实现为，设置窗口宽高比，然后设置视野深度本文默认为1000，调整相机的投影矩阵，再设置相机位置，最后执行updateProjectionMatrix函数更新相机投影矩阵。
2. 放大：放大页面需要将投影矩阵中深度系数S进行调整，本文中的放大系数为0.9，每次点击放大按钮时将页面深度变为S=0.9\*S，然后执行updateProjectionMatrix函数更新相机投影矩阵，完成放大功能。
3. 缩小：缩小页面与放大同理，只需要将系数S变为S=1.1\*S即可。

### 场景图层显/隐

场景图层的显隐控制分为两部分，一是构建场景对象映射树，另一个是点击显隐按钮时更新模型显隐状态。

构建场景对象映射树的算法实现是：初始化映射树数组和当前显示图层数组，遍历Scene内部的Children数组，首先判断数组中的节点是否为Group对象，如果是Group对象再判断是否为父节点，其次判断是否为可见，只有非父节点且可见的对象才可以加入当前显示的图形数组中，接下来继续判断当前对象是否为叶子节点，是叶子节点则加入映射树数组当中，不是叶子节点则将当前节点的Children数组进行递归处理，直至遍历完Scene下的每个子Group对象，完成映射树和当前显示对象数据的构建。

点击显隐按钮实现思路为：当点击某个图层需要显示隐藏时，当前显示对象数组会有监听变化，然后我们遍历图层所有的叶子节点与当前显示对象数组进行对比，如果叶子节点存在于当前显示对象数组中则执行showHide函数将叶子节点的visible属性设置为true处理，而如果不存在则将visible属性设为false处理，这样就能实现控制模型图层的显示与隐藏功能，具体效果如图 4‑28所示。



图 4‑28 图层显/隐效果演示

## 增材制造预处理实现

### 冗余去除和拓扑重建

冗余去除和拓扑重建，WebGL引擎渲染的STL模型数据格式是一个个无规律的三角面片，每个面片包含着三个点的坐标以及面片的法向量数据，从这种结构的数据中我们无法得知面片之间的空间拓扑信息，并且当面片的空间涵盖范围远远小于我们所需要的精度值时，这些面片对我们来说就是无意义的，因此需要进行冗余去除和拓扑重建，以便于我们下一步对模型进行切片处理。这两个步骤我们是同步进行的，具体数据处理过程为：首先提前定义三个变量resPoints（Map结构）、resEdges（Map结构）、resFaces（数组结构）分别存储点数据、边数据和面数据，再定义两个哈希生成函数pointHash和edgeHash分别对点和边进行Hash索引值的生成，然后我们开始遍历模型数据，每个面片数据获取完毕后开始计算三个点之间的距离，如果小于设定的阈值T（本文阈值为10-5），则将两个点合并为一个点，例如A点与B点，只将A点存入resPoints内，并把所有B点用A点进行替换，以此类推处理面片的第三个点，如果三个点都大于距离阈值，则再判断三个点的Z轴最大最小高度差是否也大于阈值T，如果满足则将三边与当前面数据存入resEdges和resFaces中，在存储resEdges时除了存储每条边的两个顶点，还有包含边的面片索引，方便后续寻找临接三角面片，不满足则处理下一个面片，直到所有面片被处理完毕。具体处理流程如图 4‑29所示。



图 4‑29 冗余去除与拓扑重建实现流程

### 分层切片

分层切片模块，分层切片模块接收的数据有起始切片高度、终止切片高度、层厚、以及切片绘制颜色。进入切片函数后先清除历史切片的数据缓存，再计算分层的背景图层，背景图层为一个范围正好是模型在XY轴平面投影1.5倍的半透明矩形，生成切片矩形的方法为：先通过初始切片高度和终止切片高度算出切片层数，然后循环生成PlaneGeometry对象，设置每个矩形位置再将每个PlaneGeometry实例化为Mesh存放进一个Group中，最后将整个Group加入最大的Scene对象中。生成切片背景图层后，我们要先计算出切片轮廓，计算切片轮廓的方法是以高度进行遍历，在每个高度下从resFaces中任意取一个与该高度相交且未被搜索过的面片，然后初始化轮廓点存储数组resultPoints，存储第一个面片的索引indexFirst，开始对面片中的每条边进行遍历计算交点，交点分为两种情况，如果边的一个顶点就是交点，则直接将顶点存入resultPoints中，否则计算出交点再存入resultPoints中，然后将该面打上已经搜寻过的标记，继续寻找该边的非当前临接面，继续对下一个面进行边遍历，直到下一个面的索引等于indexFirst，则结束当前封闭轮廓计算，然后继续在resFaces中任意取一个符合高度且未被搜索过的面片，直到没有符合条件的面片结束所有轮廓搜寻，具体流程如图 4‑30所示。



图 4‑30 分层切片实际处理流程

得到了每个轮廓的数据后我们还需要对轮廓进行展示，首先是将每个点进行绘制，绘制流程为：声明一个空几何体对象，再将轮廓坐标点添加到对象当中，再设置对象渲染材质，里面包含颜色和大小参数，然后将对象添加进Points点模型对象中，最后把点模型加入最大的Scene场景中执行渲染。点绘制完毕后还需要将点连接成线，绘制线流程为：将点数据转化为Vectors对象存入listPoints数组里，然后通过BufferGeometry将listPoints转化为缓冲类型几何体，设置渲染材质，然后将对象和材质放入Line线模型中，最后加入Scene场景中执行渲染。切分分层的计算和显示模块总体流程如图 4‑31所示。



图 4‑31 切片分层计算与显示流程示意图

### 轨迹填充

轨迹填充模块，轨迹填充是依据切片轮廓来对轮廓划分出的区域进行路径规划填充。本文使用的是水平路径填充算法，具体实施过程为：先输入轨迹密度，也就是每条线之间的距离，然后声明pathPoints存储轨迹路径点，然后将轮廓数组进行遍历，对每个轮廓数组重新构建存储模式，新旧轮廓存储格式如图 4‑32所示。



图 4‑32 新旧轮廓数据存储格式

先声明一个二维矩阵distinguishMatrix用来对内外轮廓进行判别，然后初始化contourInfo数组用来存储新表达形式的轮廓数据，里面包含了xMin、xMax、yMin、yMax和edgeInfo参数，代表着每个轮廓的范围以及组成轮廓的边数据，然后先对一个子轮廓数据进行遍历，遍历的过程中就能得到每个轮廓的范围极值和组成的边信息，并存入contourInfo数组中；经过上面步骤我们得到了contourInfo数组，然后使用双层循环遍历contourInfo数组得到内外轮廓判别矩阵distinguishMatrix，在循环中将外层循环的当前轮廓与内层循环的所有轮廓范围进行范围对比，如果当前轮廓有被其他轮廓包含则将判别矩阵对应位置加1并把那个大的轮廓索引值和X轴范围最小值存入判别矩阵，最后得到的distinguishMatrix数据就存放着内外轮廓判别依据。接下来我们要利用distinguishMatrix判别矩阵来对轮廓进行合并操作，开始循环判别矩阵，如果轮廓的判别矩阵对应位置的存储数组长度为偶数则为外轮廓，奇数则为内轮廓，如果是外轮廓则无需处理，是内轮廓则需要寻找到包含它的轮廓中最接近它的轮廓，并将该内轮廓的边数据合并到最接近轮廓的边数据中，同时将该内轮廓的数据从contourInfo中删除，内外轮廓判别过程如图 4‑33所示。



图 4‑33 内外轮廓判别过程

当得到处理后的轮廓线数据，我们通过水平线扫描算法就可以得到轨迹规划的路径了。具体实施流程为：初始化扫面线初始高度和截止高度分别为轮廓在Y轴的最大值和最小值，然后遍历每个轮廓区域从初始高度开始向下扫描，以轨迹密度递增，扫描过程中把每个Y轴高度与当前轮廓线求交点，如果当前交点处于轮廓边之间，则将该交点存入交点数组内，如果该交点在两边的最低点则将该交点存入两次交点数组，如果交点位于两边的最高点则跳过该交点，由此得到若干个交点后，按照X轴方向从小到大排序，最后每两个点进行连线（例如ABCDEF这六个点，得到的轨迹就是A-B、C-D、E-F这三条线段）得到最后的轨迹路径； 水平扫描轨迹填充具体流程如图 4‑34所示。



图 4‑34 水平扫描轨迹填充具体流程

### G代码生成

GCode生成，通常3D打印机使用的GCode文件主要由三部分组成，分别为：初始化部分、打印部分、结尾部分。初始化部分主要是设置一些准备参数，比如复位打印喷头和平台的位置，挤出头和打印平台的预热温度，挤出机计数置零等。初始化部分的具体实例化过程为：设置坐标单位、设置喷头和机床温度、喷头和打印平台还原、挤出长度归零、移动喷头到起始位置。打印部分则是对轨迹路径进行转码显示，用来完成实体打印，需要关注的是打印头的移动速度和喷头送丝量，同时还要根据轨迹的断连控制打印头的升降和送丝的间断。结尾部分的G代码就是完成打印后对打印机进行关闭的操作，比如复位喷头和打印平台，停止加热材料，关闭驱动电机电源等。G代码在不同设备和软件的下的内容结构都有所不同，所以本文中G代码的初始化和结尾部分较为固定，如果有定制化需求，用户可以将G代码自行下载修改。

下面分别给出初始化部分、打印部分和结尾部分的简要实例代码。

1. 初始化部分。

G21; 将单位设置为毫米

G90; 采用绝对坐标

M104 S205; 设置挤出头温度为205℃

G28; 喷头和底部平台回归原点

G1 Z5 F5000 E1; 沿Z轴移动5mm，速度为3000mm/min，并挤出1mm的丝材

M109 S205; 等待喷头温度到达205℃

G92 E0; 打印头丝材挤出量归零

M82; 打印头采用绝对坐标

②打印部分。

G0 X67.263 Y59.881 Z0.500 F7800.000; 打印头以7800mm/min的速度从原点运动到坐标(67.263,59.881,0.500)处，不进行送丝

G1 X68.176 Y59.025 E1 F1080.000; 打印头从上个点以1080mm/min的速度移动到坐标(68.176, 59.025,0.500)处，同时匀速送丝1mm的长度进行打印

G1 X 68.709 Y 58.616 E2 F1080.000; 打印头从上个点以1080mm/min的速度移动到坐标(68.709, 58.616,0.500)处，同时匀速送丝1mm的长度进行打印

③结尾部分。

G92 E0; 打印头丝材挤出量归零

M104 S0; 停止加热打印头

M140 S0; 停止给平台加热

G28 X0 Y0 Z0; 将打印头回归原点

M84; 电机断电

### 动画模拟

动画模拟填充轨迹，我们需要利用Three.js中的帧动画来实现一个标记球体沿着填充轨迹进行运动，首先需要定义一个SphereGeometry的球体对象，然后设置球体的大小于材质，然后定义一个CatmullRomCurve3的3D样条曲线，再用水平扫描填充计算出来的数据对该曲线进行初始化，接下来用Float32Array类型数组来生成动画的时间序列，再利用轨迹路径坐标去生成一个和时间序列相对于的位置坐标序列，这样我们就能得到每个关键帧的时值与当前时值运动到的坐标点，再使用Three.js提供的KeyframeTrack函数生成动画轨迹数据，然后使用AnimationClip和AnimationMixer去将球体与路线轨迹进行绑定，最后调用play方法去执行动画。具体实现代码如图 4‑35所示。



图 4‑35 动画模拟代码

### 日志记录

日志可以看做是关键操作数据埋点，可以记录用户在增材制造预处理过程中所产生的数据，为后续结果分析提供参考。因为本系统是基于组件模块化开发，要对不同组件间执行的结果进行存储，需要涉及到组件通信问题，Vuex库为Vue架构代码提供了页面数据共享的解决方案，我们首先在store文件中定义一个loggingData.js文件，该文件内部分为State、Mutations、Actions，State用来存放日志总数据，我们将格式定义{action——操作说明,date——记录时间}；Mutations内定义同步方法，用来修改State中的数据，我们暂时定义了添加新日志记录和重置日志记录这两个函数；Actions中也是定义一些操作函数来对State中的数据进行修改，但是函数为异步执行，由于本系统中还没有需要异步记录的数据，因此暂不使用Actions。详细代码如图 4‑36所示。



图 4‑36 日志记录Vuex定义

## 数据库设计与实现

本文数据库使用的是非关系型数据库MongoDB，开发是使用Mongoose框架来进行实现。本系统所涉及到的数据集合有两类，一个是用户信息，一个是模型信息，在Mongoose中是使用Scheme结构去定义数据模型，下面是用户信息Scheme与模型数据Scheme的详细设计。

### 用户信息Scheme

用户信息集合需要存储用户名、真实姓名、密码、年龄、单位、专业方向、可用模型索引、用户角色、用户注册时间这些字段，详细数据类型如表 4‑3所示。

|  |  |  |
| --- | --- | --- |
| 字段 | 参数 | 类型 |
| 用户名 | username | {type: String, default: 'user'} |
| 真实姓名 | realname | {type: String, default: '暂无'} |
| 密码 | password | {type: String, default: '123456'} |
| 年龄 | age | {type: Number, default: 0} |
| 单位 | company | {type: String, default: '暂无'} |
| 专业 | major | {type: String, default: '暂无'} |
| 可用模型 | usableModel | {type: mongoose.Schema.Types.ObjectId} |
| 角色 | role | {type: String, default: 'user' or 'admin'} |
| 注册时间 | date | {type: Date, default: Date.now} |

表 4‑3 用户信息Scheme

### 模型数据Scheme

模型数据集合需要存储模型藐视、模型图片地址、模型文件地址、模型名字、模型所有人ID、是否公共模型、模型入库时间，详细数据类型如表 4‑4所示。

|  |  |  |
| --- | --- | --- |
| 字段 | 参数 | 类型 |
| 模型描述 | modelDesc | {type: String, default: '暂无'} |
| 模型图片地址 | modelImgUrl | {type: String, default: ''} |
| 模型文件地址 | modelFileUrl | {type: String, default: ''} |
| 模型名字 | modelTitle | {type: String, default: '暂无'} |
| 模型所有人ID | ownerId | {type: String, default: ''} |
| 是否公共模型 | isPublic | {type: String, default: false} |
| 注册时间 | date | {type: Date, default: Date.now} |

表 4‑4 模型数据Scheme

## 服务端设计与实现

服务端我们使用到的是Express.js框架，可以快速搭建小型灵活的后台服务，系统中后端服务主要涉及到登录验证、用户信息处理和模型信息处理。

### Token认证

由于前后端分离模式，登录校验和权限接口认证需要依赖token机制来实现，本文使用的是基于JWT的认证方案，具体实现如下：

1. 用户在登录页面提交用户名、密码。
2. 服务端获取用户名、密码，然后通过数据库查询校验。
3. 服务端生成Token返回给前端，本文的Token生成规则是HS256。
4. 前端获取到Token后将Token保存在localStorage中。
5. 前端在之后凡是涉及到权限的请求接口时会将Token放在请求的头部中。
6. 后端接收到前端返回的Token时会进行解码，得到Header和Payload，然后通过秘钥来加密生成Token以此检验Token是否有效。

基于JWT的Token认证实现流程如图 4‑37所示。



图 4‑37 基于JWT的Token认证实现流程

### 用户信息接口服务

前端对用户信息进行操作，需要服务端提供以下接口：登录接口、注册接口、获取用户信息接口、更新用户信息接口、获取用户列表接口、删除用户接口。详细接口设计如表 4‑5所示：

|  |  |  |
| --- | --- | --- |
| 接口说明 | 字段 | 参数 |
| 登录接口 | /tokens/login | 用户名、密码 |
| 注册接口 | /tokens/register | 用户名、真实姓名、密码 |
| 获取用户信息接口 | /tokens/userInfo | 用户ID |
| 更新用户信息接口 | /tokens/updateUserInfo | 用户ID、用户信息 |
| 获取用户列表接口 | /tokens/userList | 用户名OR真实姓名 |
| 删除用户接口 | /tokens/deleteUser | 用户ID |

表 4‑5 用户信息接口服务详情

### 模型数据接口服务

模型数据信息操作需要服务端提供下列接口：上传文件接口、获取模型列表接口、新增模型接口、删除模型接口、更新模型接口、公共模型使用接口、模型移除可使用接口。详细接口设计如表 4‑6所示：

|  |  |  |
| --- | --- | --- |
| 接口说明 | 字段 | 参数 |
| 上传文件接口 | /model/upload | 用户ID、文件数据 |
| 获取模型列表接口 | /model/getModelList | 用户ID |
| 新增模型接口 | /model/userInfo | 用户ID、模型信息 |
| 删除模型接口 | /model/deleteModel | 用户ID、模型ID |
| 更新模型接口 | /model/updateModel | 模型ID、模型信息 |
| 公共模型使用接口 | /model/addUsableModel | 模型ID、用户ID |
| 模型移除可使用接口 | /model/removeUsableModel | 模型ID、用户ID |

表 4‑6 模型数据接口服务详情

## 系统测试

### 服务器部署

基于Web的增材制造预处理系统的服务器部署，本文使用UCloud云主机，型号是AMD版快杰型，系统盘为40G RSSD云盘，BGP带宽为1M。服务器部署分为前端、后端、数据库三部分。前端部署：前端将访问后端的地址修改为服务器的公网IP，再通过npm run build命令将开发环境代码打包成生产环境代码，然后将生产环境代码部署在服务器的8080端口上；后端则将后端代码打包推送到云服务器上，再通过pm2工具启动后端服务；数据库则是在服务器上安装MongoDB并配置数据存储路径，再将本地数据服务器启动。本文的云服务器公网IP为106.75.190.235，因此通过网址http://106.75.190.235:8080即可访问应用。

### 功能测试

1. 登录注册：用户名myTest，真实姓名测试号，密码123456.



图 4‑38 登录注册

1. 上传模型：模型名称螺旋桨，模型描述螺旋桨，设置权限公开，上传图片螺旋桨.png，上传模型螺旋桨.stl.



图 4‑39 上传模型

1. 修改模型：模型描述修改为螺旋桨修改，设置权限修改为私有。



图 4‑40 修改模型

1. 删除模型。



图 4‑41 删除模型

1. 所有模型库中公共模型加入我的模型库。



图 4‑42 公共模型加入我的模型库

1. 个人中心。



图 4‑43 个人中心

1. 选择模型进入工作台页面。



图 4‑44 模型进入工作台页面

1. 分层切片：起始切片高度-5，终止切片高度5，层厚1，绘制颜色0xff0000.



图 4‑45 分层切片

1. 轨迹生成：轨迹密度为1，绘制颜色为0xffffff.



图 4‑46 轨迹生成

1. 图层显示与隐藏。



图 4‑47 图层显示与隐藏

1. G代码生成与下载。



图 4‑48 G代码生成与下载

1. 动画模拟轨迹。



图 4‑49 动画模拟填充轨迹

1. 日志记录。



图 4‑50 日志记录

1. 用户信息管理。



图 4‑51 用户信息管理页面



图 4‑52 修改用户和查询用户

1. 模型信息管理



图 4‑53 模型信息管理

在功能测试中，针对系统的6个模块编制了功能测试用例表，再对每个模块中的功能项进行测试，整理出我们的测试用例表，如表 4‑7所示。

|  |  |  |
| --- | --- | --- |
| 模块 | 测试内容 | 实际结果 |
| 注册登录模块 | 注册系统账号并登录系统 | 符合预期 |
| 用户模型管理模块 | 用户对模型数据进行上传、修改、删除、共享 | 符合预期 |
| 模型显示交互模块 | 定义模型展示区域，对视窗进行还原、放缩操作，控制模型数据、参考坐标、包络盒、分层切片、轨迹的图层显隐，还有模拟轨迹动画 | 符合预期 |
| 增材制造预处理模块 | 对模型数据进行拓扑重建、分层切片、轨迹规划处理 | 符合预期 |
| 系统用户信息管理模块 | 对系统所有用户信息进行搜索、删除、修改 | 符合预期 |
| 系统模型管理模块 | 对系统所有模型数据进行搜索、删除、修改 | 符合预期 |

表 4‑7 系统功能测试内容表

### 性能测试

性能测试使用的电脑配置为：Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz处理器、8GB RAM和NVIDIA GeForce GTX 1050 Ti显卡，并在Google Chrome 89.0.4389.90版本上运行。

1. 增材制造预处理是本系统的主要功能，因此我们需要对每一步的处理算法进行测

试，判断在线应用程序的直观标准是用户等待时间，一般用户可接受的Web程序等待时间为5s，如果在某个算法的处理过程中用时超过了5s，则说明该系统的用户体验有待提高;判断在线三维处理应用还要参考刷新率指标，当实际使用中刷新率在30fps到60fps之间就是人眼可接受的范围。我们下面使用不同的模型来对算法进行测试，拓扑重建、分层切片、轨迹填充的测试结果如下表所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 拓扑重建算法 |  |  |  |
| 模型名称 | 面片数量(个) | 刷新率(FPS) | 实际用时(MS) |
| 棱形五角星 | 60 | 60 | 6 |
| 轮轴 | 19898 | 60 | 154 |
| 水杯 | 68730 | 56 | 338 |
| 兔子 | 75710 | 52 | 457 |
| 旋转城堡 | 219828 | 50 | 1118 |

表 4‑8 拓扑重建算法测试

|  |  |  |  |
| --- | --- | --- | --- |
| 分层切片算法 |  |  |  |
| 分层密度为1 |  |  |  |
| 模型名称 | 分层数量(层) | 刷新率(FPS) | 实际用时(MS) |
| 棱形五角星 | 26 | 60 | 188 |
| 轮轴 | 20 | 51 | 362 |
| 水杯 | 100 | 45 | 2772 |
| 兔子 | 20 | 54 | 239 |
| 旋转城堡 | 24 | 55 | 1148 |

表 4‑9 分层切片算法测试

|  |  |  |  |
| --- | --- | --- | --- |
| 轨迹填充算法 |  |  |  |
| 分层密度为1 | 填充密度为1 |  |  |
| 模型名称 | 轮廓数量(个) | 刷新率(FPS) | 实际用时(MS) |
| 棱形五角星 | 26 | 60 | 57 |
| 轮轴 | 60 | 49 | 648 |
| 水杯 | 236 | 40 | 2543 |
| 兔子 | 31 | 56 | 46 |
| 旋转城堡 | 83 | 53 | 83 |

表 4‑10 轨迹填充算法测试

通过上面的测试结果得出随着面片数量增大以及模型复杂度的增加，计算机消耗的计算资源会逐渐增加，但都在用户的可接受范围之内，并且根据调研显示大部分用于3D打印成型的模型文件数据量都在10万面片以下，因此该系统可以满足增材制造预处理的算法需求。并且由于算法消耗的是客户端环境的计算能力，如果用户的机器配置越高，那么系统的流畅度就会越高。

1. 系统高并发测试采用Python的多线程去模拟并发效果，向服务器施加并发访问

压力，检测系统在并发压力下响应接口的能力。在Python并发测试代码中计算错误请求个数以及HTTP平均响应时长，同时使用pm2提供的monitor性能监视器来监听CPU占用率等情况。

高并发测试的模拟请求为1000个，只要系统能在1000个请求压力情况下还能较为流畅的进行功能的使用并且服务器主机的CPU占用率不超过50%，就说明该系统满足实际运行场景需求。我们将初始并发访问数量设置为200，每间隔10秒增加200个并发用户，直到模拟到1000个并发用户数为止，同时对服务器的性能进行监听。上述测试结果经整理如表 4‑11所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 并发数 | 错误请求数(个) | HTTP平均响应时长(MS) | CPU占用率(%) |
| 200 | 0 | 11ms | 31 |
| 400 | 0 | 12ms | 30 |
| 600 | 0 | 13ms | 34 |
| 800 | 0 | 14ms | 32 |
| 1000 | 0 | 12ms | 35 |

表 4‑11 并发测试结果



图 4‑54 并发测试CPU占用率

根据上面的测试结果我们可以看出系统对请求的响应时间在13ms左右，因为服务端CPU的压力在承载值之内的时候，只要网络环境的波动不是太大，那么HTTP请求的响应时间一般都不会有太大变化，而且随着并发量的上升，服务器主机的CPU使用率只是稍微增加，完全可以满足实际生产需求。

## 本章小结

本章对增材制造预处理Web平台的实现进行了详细的阐述和展示，并对系统进行功能与性能测试，从测试结果来看，系统满足预期开发的要求。

# 总结与展望

## 本文主要内容及结论

本文对增材制造预处理使用到的相关软件进行调研，并与目前流行的Web应用进行比较分析，得出基于Web的应用在未来更有发展潜力。同时我们还通过对3D打印预处理技术原理的分析，确定了系统的几个核心过程，并使用Vue、Three.js、Node.js、Express.js和MongoDB开发了一个即时运行、便捷高效、数据可控的增材制造预处理Web应用。主要完成了以下工作：首先针对JavaScript语言的内置数据结构特性设计了3D打印预处理的几个核心模块算法，主要包括STL模型的导入导出、拓扑重建、分层切片、轨迹填充、生成G代码，并将这些模块封装成黑盒函数提供给程序调用；然后调用WebGL图形接口，使用二次封装的Three.js框架去完成模型数据的显示与交互操作，针对一些实际业务需求，比如放缩还原，坐标轴和底部网格的绘制，场景图层的显示与隐藏等，自行设计算法实现；然后基于前后端分离的开发模式去对整个系统的功能进行实现，包括前端页面实现、后端接口服务实现和数据库集合实现；最后对该Web应用进行了相关测试，实验结果表明：系统用户体验良好，功能实现完整，算法准确可行，应用界面美观稳定，基本可以满足增材制造数据预处理的需求。

## 本文的主要创新点

本文通过对现有3D打印预处理软件进行调研分析，深入研究模型在预处理流程中的算法，并结合Web平台的优势，设计并开发出基于Web 的增材制造预处理平台，主要创新点如下：

1. 将增材制造预处理相关软件的算法进行Web端迁移，把算法在原有的基础上

根据JavaScript语言的数据结构特性进行改进，使算法满足在Web端实时处理的要求，并结合近几年才发展起来的WebGL技术，流畅美观的实现了模型数据的展示与交互。

1. 严格按照前后端分离原则，把用户管理、模型管理与3D打印模型数据处理

整合到一个Web应用中。并且由于Web端的高效3D处理技术在近几年才慢慢发展起来，所以目前市面上还没有3D打印数据处理Web平台的出现，本文工作对未来3D打印在Web端的实践有一定的开拓性意义。

## 展望

由于增材制造预处理软件大多数是客户端，开发成本高，无法跨平台还容易被破解，随着3D打印技术的普及，未来相关应用的形态肯定是能跨平台即开即用，且要有一定的数据管理能力，而Web应用正好可以满足上述需求，并且网页的三维开发技术变得越来越成熟，基于此背景下，本文是对3D打印数据处理应用在Web上的一个探索实践，距离真正成熟的商业化系统，仍有以下几点改进方面：

1. 由于系统研究的对象是STL模型数据，我们只会关注模型是否按照流程被

完整处理，而在实际打印过程中我们需要考虑打印的工艺，可能需要添加支撑元素，或者需要使用一些其他切片方式，这都需要跟进开发相应的处理算法。

1. 系统的交互方式还可以再进行优化，暂时很多场景的背景灯光，模型的材质

颜色都是定死了，如果需要更自由的展现方式，还需要针对每个场景中的对象进行相关配置开发。

1. 目前系统处理的模型暂时只是中小模型，如果涉及到特别大型的模型数据，

还可以设计合适的算法使用多个服务器资源进行同步计算，做到计算资源均衡，甚至可以使用新系统架构开放的API去实现计算均衡，例如华为研发的鸿蒙系统就可以实现不同设备之间均衡计算资源。

1. 由于系统带有数据管理的性质，具备一定的用户黏性，在未来可以考虑加入

更多丰富的功能，例如模型的版权认证，用户团队创建，移动端适配等。

# 致 谢

# 

# 参考文献

武汉光电国家研究中心规定：硕士学位申请人的文献阅读量不少于600篇（高于学校规定的40篇），其中外文文献不得少于2/3；近五年的论文不得少于1/3；绪论部分应对所读文献加以分析和综合。

**参考文献**

1. G. K. Awari et al. Additive Manufacturing and 3D Printing Technology: Principles and Applications[M]. CRC Press, 2021
2. M. Chalfie, S. R. Kain. Green fluorescent protein: properties, applications, and protocols. Hoboken, New Jersey: Wiley-interscience, 1998
3. Tosto Claudio et al. Fused Deposition Modelling (FDM): New Standards for Mechanical Characterization[J]. Macromolecular Symposia, 2021, 395(1)
4. 卢秉恒,李涤尘.增材制造(3D打印)技术发展[J].机械制造与自动化,2013,42(04):1-4.
5. Wohlers Associates, Wohlers Report 2013--Additive manu-facturing and 3D printing state of the Industry Annual WorldwideProgress Report, 2013, USA
6. 陈志茹,夏承东,李龙,等.3D 打印材料[J].金属世界,2018(05):15-18.
7. 谢越. 面向普通用户的三维模型设计方法研究[D].浙江大学,2016.
8. Sutherland I E. Sketch pad a man-machine graphical communication system[C]. Proceedings of the SHARE design automation workshop. ACM. 1964:6-329.
9. Requicha A A, Voelcker H B. Constructive solid geometry[J]. 1977.
10. Requicha A A, Voelcker H B. Solid modeling: Current status and research directions[J]. Computer Graphics and Applications, IEEE, 1983, 3(7):25-37.
11. Requicha A A, Voelcker H B. Boolean operations in solid modeling: Boundary evaluation and merging algorithms[J]. Proceedings of the IEEE, 1985, 73(1):30-44.
12. Braid I C. The synthesis of solids bounded by many faces[J]. Communications of the ACM, 1975, 18(4):209-216.
13. Braid I. On storing and changing shape information[C]. ACM SIGGRAPH Computer Graphics. Vol 12. ACM. 1978:252-256.
14. Braid I, Hillyard R. Geometric modeling in ALGOL 68[J]. ACM Sigplan Notices, 1977, 12(6):168-174.
15. 阿占文.家庭化3D打印机分层软件的设计与实现[D].华中科技大学, 2014.
16. 丘宏扬,陈松茂,刘斌等.快速成形系统中 STL 文件切片算法的研究[J].锻压技术,2005,30(4):35-39.
17. Neil Savage. Weaving the web[J]. Communications of the ACM, 2017, 60(6): 20-22.
18. Tabarés Raúl. HTML5 and the evolution of HTML; tracing the origins of digital platforms[J]. Technology in Society, 2021, 65.
19. Aiello M. The Browser Lament[J]. In: The Web Was Done by Amateurs. Springer, Cham, 2018.
20. Claussnitzer, Melina, Cho, Judy H, Collins, Rory, et al. A brief history of human disease genetics[J]. Nature, 2020, 577(7789), 179–189.
21. Jesse James Garrett. Ajax: A new approach to web applications[EB/OL]. 2005, https://www.adaptivepath.com/publications/essays/archives/000385print.php.
22. Stian Reimers, Neil Stewart. Adobe Flash as a medium for online experimentation: A test of reaction time measurement capabilities[J]. 2007, 39(3), 365–370.
23. G. Richards, S. Lebresne, B. Burg, and J. Vitek. An analysis of the dynamic behavior of JavaScript programs. In: ACM Sigplan Notices, 2010, vol. 45, no. 6, pp. 1–12.
24. J. Resig and others. jquery: The write less, do more, javascript library. Disponvelem, 2009.
25. Tim Hesterberg[J]. Bootstrap, 2011, 3(6), 497–526.
26. Tilkov S, Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs[J]. IEEE Internet Computing, 2010, 14(6):80-83.
27. 陆凌牛. HTML5与CSS3权威指南[M]. 机械工业出版社, 2011:60-100.
28. S Mohorovi ci c. Implementing Responsive Web Design for Enhanced Web Presence[C]. Information & Communication Technology Electronics & Microelectronics (MIPRO), 2013 36th International Convention on IEEE 2013:1206-1210.
29. 杜艳美, 黄晓芳. 面向企业级web应用的前后端分离开发模式及实践[J]. 西南科技大学学报, 2018, v. 33; No.130(02):86-90.
30. Chandler, Jennifer Sara. Techniques for Visualizing Particle-Based Flow Fields[D]. ProQuest Dissertations and Theses Full-text Search Platform, 2016.
31. 侯聪聪,南琳,张磊等.基于分组的STL模型快速切片算法[J].制造业自动化,2014,(9):12-15.
32. 巢海远,刘景,童晶等.一种处理带有边界的非封闭STL模型的切片算法[J].计算机集成制造系统, 2015,21(10):2587-2595.
33. Gan W, Zhou Y H. An improved method of hash table based on transform and conquer[J]. Information and Computational Science, 2012, 9(17): 5361－5371.
34. Hao J B Fang L, Yang H F. An Improved Boundary Extraction Method of STL Model Based on Edge Curvature Estimation[J]. Computer Modeling and New Technologies, 2014, 18(10):252-258.
35. 何泳江.一种工业CT切片直接生成3D打印G代码的方法研究[D].重庆大学,2019.
36. 徐文鹏,王伟明,李航等.面向3D打印体积极小的拓扑优化技术[J].计算机研究与发展,2015,52(1):38-44.
37. 侯章浩,乌日开西·艾依提.3D打印的路径规划研究综述[J].机床与液压,2016,44(5):179-182.
38. 张龙,李旭东,郭德昌等.3D打印过程的计算机仿真[J].计算机仿真,2014,31(8):226-229,300.
39. Freeman A. Extending Vue.js[M]. Pro Vue.js 2. 2018.
40. Yuanyi Chen and Chen Yuanyi. Application of Web Program Development Based on Struts Framework[J]. Journal of Physics: Conference Series, 2020, 1648(3) : 032193-.
41. Kouraklis J. MVVM as design pattern[M]. MVVM in Delphi.Berkeley, CA: Apress,2016:1-12.
42. Nazarov R, Galletly J. Native browser support for 3D rendering and physics using WebGL, HTML5 and JavaScript[C]. Communications of the ACM. 2013.
43. IOANNIS K, KYRIAKOS-IOANNIS D, NIKOLAOS D. Is Node.js a viable option for building modern web applications? A performance evaluation study[J]. Computing, 2015, Vol.97(10): 1023-1044.
44. MCMANUS S. Write Maintainable Web Apps With Express[J]. Net, 2014(Mar.TN.251).
45. 范凯.NoSQL数据库综述[J].程序员,2010(6):76-78.
46. Paolo Atzeni et al. Data modeling in the NoSQL world[J]. Computer Standards & Interfaces, 2020, 67.
47. Sangeeta Gupta. Comparative Analysis of Nosql Specimen with Relational Data Store for Big Data in Cloud[J]. International Journal of Distributed and Cloud Computing, 2015, 3(1):17-23.
48. TVD Maßen, H Lichter. Modeling Variability by UML Use Case Diagrams. in: Proceedings of International Workshop on Requirements Engineering for Product Lines, IEEE Joint International Requirements. Engineering Conference(RE02), Essen, Technical Report: ALR-2002-033, 2002, 19-25.
49. G Booch, J Rumbaugh, Ivar Jacobson. UML 用户指南, 第 2 版. (修订版). 北京:人民邮电出版社, 2013(1): 258-283.